



INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

CIC

Tesis

**“NAVEGACIÓN DE UN ROBOT HUMANOIDE MEDIANTE
REDES DE PETRI Y LÓGICA DIFUSA”**

Que para obtener el título de

“Maestro en Ciencias de la Computación”

Presenta

Federico Furlán Colín

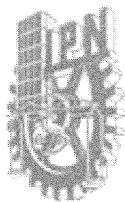
Asesores

Dr. Juan Humberto Sossa Azuela

Dra. Elsa Rubio Espino



MÉXICO, D.F. , JUNIO 2013



INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 18:00 horas del día 18 del mes de junio de 2013 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Navegación de un robot humanoide mediante redes Petri y lógica difusa"

Presentada por el alumno:

Furlán

Apellido paterno

Colín

Apellido materno

Federico

Nombre(s)

Con registro:

B	1	1	0	8	4	2
---	---	---	---	---	---	---

aspirante de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron **APROBAR LA TESIS**, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de Tesis


Dr. Juan Humberto Sossa Azuela


Dra. Elsa Rubio Espino


Dr. Herón Molina Lozano


Dr. Victor/Hugo Ponce Ponce


Dr. Miguel Santiago Suárez Castañón

PRESIDENTE DEL COLEGIO DE PROFESORES




Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN
EN COMPUTACIÓN
DIRECCIÓN



INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 27 del mes de junio del año 2013, el que suscribe **Federico Furlán Colín** alumno del Programa de Maestría en Ciencias de la Computación con número de registro **B110842**, adscrito al Centro de Investigación en Computación, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección del **Dr. Juan Humberto Sossa Azuela** y la **Dra. Elsa Rubio Espino** y cede los derechos del trabajo titulado "*Navegación de un robot humanoide mediante redes Petri y lógica difusa*", al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección fedfurlan@gmail.com, hsossa@cic.ipn.mx ó erubio@cic.ipn.mx. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Federico Furlán Colín

Resumen

Se implementó un sistema de navegación en un robot humanoide con la finalidad de desplazarse en un entorno limitado para localizar un objetivo, evitando colisiones con el entorno. Se propuso utilizar lógica difusa conjuntamente con una herramienta para el modelado de eventos discretos, en la parte de lógica difusa se eligió utilizar un sistema difuso de toma de decisiones o FDMS por sus siglas en inglés y para realizar el modelo a eventos discretos de las acciones y tareas del robot se utilizaron las redes de Petri.

Para ello se utilizó un robot humanoide BIOLOID Premium con el cual se realizaron los experimentos, se le montó una cámara HAVIMO 2.0 para localizar el objetivo así como un arreglo de tres sensores infrarrojos para medir distancia. Los sensores están posicionados al frente y a ambos lados del robot para monitorear el entorno en el cual se desplaza. Dadas las limitadas capacidades del robot para procesar información y realizar tareas paralelas fue necesario establecer una comunicación entre él y una computadora mediante el protocolo Zigbee para desarrollar el sistema de navegación.

Para localizar el objetivo, el robot realiza un paneo de la cámara de casi 360°, en el cual la cámara HAVIMO detecta si el color del objetivo buscado está dentro del rango de visión, si el robot lo encuentra trata de posicionarse en línea recta a él y comienza a acercarse hasta alcanzar una distancia muy próxima y entonces finaliza la tarea, de lo contrario es necesario explorar el entorno para localizar el objetivo, donde el sistema difuso de toma de decisiones se encarga de elegir qué movimiento realizará el robot dependiendo de la información de los sensores infrarrojos, entre las posibles acciones están caminar o girar. La interacción entre las tareas del robot y las posibles acciones que dependiendo de los valores de los sensores, son modeladas usando redes de Petri (rdP). El uso de las rdP nos permite monitorear el comportamiento del robot y tener un buen desempeño al localizar el objetivo. Asimismo, las RdP también son una herramienta de análisis para prevenir conflictos en las acciones del robot.

Abstract

A navigation system was implemented in a humanoid robot in order to move inside a limited environment to find a target, while avoiding collisions with obstacles or walls. It was proposed to use fuzzy logic in conjunction with a modeling tool for discrete events systems. For the fuzzy logic part, a fuzzy decision making system was selected and, to model the actions and task of the robot the Petri nets was used.

A Bioloid Premium humanoid robot was employed to perform the experiments, a camera HAVIMO 2.0 was mounted on the robot to locate the target as well as an array of three infrared distance measure sensors, which were positioned at the front and at the sides of the robot to get information of the environment where the robot moves. According to the limited capabilities of the robot to process information and perform parallel task, it was necessary to establish a communication between the robot and a computer using Zigbee protocol to develop the navigation system.

In order to locate the target, the robot perform a panning movement of the camera of almost 360 ° in which HAVIMO camera detects if it is the color of the target within viewing range. If it is found, the robot tries to get a position in a straight line to the target and starts approaching until a small distance is reached and then the task is completed. Otherwise, it is necessary to explore the environment to find the target, in this case the fuzzy systems is responsible to choose which movement will be done by the robot. It is depending on the infrared sensors information, the possible actions are walking or turning. The interaction between robot tasks and possible actions depending on the values of the sensors are modeled using Petri nets (PN), in order to observe the behavior of the robot and to have a well performing while reaching the target, the PN are also an analysis tool to prevent conflicts in the actions of the robot.

Agradecimientos

Agradezco el apoyo económico otorgado para la realización de este trabajo de tesis, esto en el marco de los proyectos con número de registro: SIP20121311, SIP20131182, SIP20131505 y CONACYT 155014.

Agradezco a mis asesores el Dr. Juan Humberto Sossa Azuela y a la Dra. Elsa Rubio Espino, al comité revisor de la tesis, al CONACYT por la beca que me otorgó para realizar la maestría, a mis compañeros del centro de investigación en computación que me apoyaron para terminar esta tesis, a Carolina Melchor por ayudarme con la ortografía, a Dios, a mi familia y a Leticia Ruiz Vázquez .

Índice

Resumen.....	- 3 -
Abstract	- 4 -
Agradecimientos	- 5 -
Índice de figuras	- 8 -
Índice de Tablas.....	- 10 -
Capítulo 1	- 12 -
Introducción	- 12 -
1.1.-Los robots humanoides hoy en día (1).....	- 12 -
1.2.-La navegación en robots móviles.....	- 15 -
1.3.-Las redes de Petri (9)	- 16 -
1.4.- Estado del Arte.....	- 17 -
1.5.-Justificación.....	- 21 -
1.6.-Solución propuesta	- 22 -
1.7.-Objetivos	- 23 -
1.7.1.-Objetivo general.....	- 23 -
1.7.2.-Objetivos específicos	- 24 -
Capítulo 2	- 25 -
Marco Teórico	- 25 -
2.1.-Redes de Petri (9) (15)	- 25 -
2.1.1.-Lugares, transiciones y arcos	- 25 -
2.1.2.-Marcaje	- 26 -
2.1.3.-Disparar una transición.....	- 27 -
2.1.4.-Características esenciales	- 28 -
2.1.5.-Propiedades de las redes de Petri.....	- 29 -
2.1.6.-Invariantes	- 32 -
2.1.7.-Buscando las propiedades de las redes de Petri.....	- 33 -
2.2.-Lógica difusa (16) (11) (6).....	- 39 -
2.2.1.-Conjuntos difusos.....	- 41 -
2.2.2.- Características de las funciones de membresía	- 42 -

2.2.3.- Fusificación.....	- 43 -
2.2.4.- Asignación de funciones de membresía	- 43 -
2.2.5.- Defusificación.....	- 46 -
2.2.6.- Sistemas difusos de inferencia.....	- 49 -
2.2.7.- Métodos difusos de inferencia	- 51 -
2.2.8.- Método difuso de inferencia Mamdani	- 51 -
Capítulo 3	- 54 -
Plataforma de Prueba	- 54 -
3.1.-Bioloid Kit Premium (17).....	- 54 -
3.2.-Módulo ZigBee (18).....	- 59 -
3.3.-Motor Dynamixel AX-12.....	- 60 -
3.4.-HAVIMO 2.0	- 62 -
3.4.1.- Algoritmo de regionalización	- 64 -
3.5.-Sensor infrarrojo de distancia.....	- 65 -
Capítulo 4	- 66 -
Desarrollo.....	- 66 -
4.1.-Ensamblado del robot Bioloid.....	- 66 -
4.2.-Montaje de la cámara y los sensores.....	- 67 -
4.3.-Linealización de los sensores	- 70 -
4.4.-Calibración de la cámara.....	- 74 -
4.5.-Cálculo de distancia mediante la cámara	- 76 -
4.6.-Movimientos del robot para caminar y girar	- 78 -
4.7.-Algoritmo para ubicar el objeto y posicionar el robot.....	- 80 -
4.8.-Sistema difuso de toma de decisión (Sistema difuso de inferencia)	- 82 -
4.8.1.- Conjuntos difusos de entrada	- 83 -
4.8.2.- Conjunto difuso de salida.....	- 87 -
5.8.3.- Definición de las reglas difusas	- 88 -
4.8.4.- Implementación del sistema difuso	- 91 -
4.9.-Red de Petri.....	- 92 -
4.9.1.- Ecuación de la red de Petri	- 94 -
4.9.2.- Propiedades de la red de Petri.....	- 98 -
4.10.-Interfaz	- 103 -

Capítulo 5	- 108 -
Resultados experimentales y análisis.....	- 108 -
5.1.-Área de trabajo	- 108 -
5.2.-Configuración 1	- 110 -
5.3.- Configuración 2	- 111 -
5.4.- Configuración 3	- 114 -
5.5.- Configuración 4	- 116 -
Capítulo 6	- 119 -
Conclusiones, trabajos futuros y recomendaciones.	- 119 -
6.1.- Conclusiones	- 119 -
6.2.- Trabajo futuro	- 120 -
6.3.- Recomendaciones	- 121 -
Referencias.....	- 122 -
Anexos	- 124 -
Biblioteca para comunicación ZigBee.	- 124 -
Código de la interfaz	- 125 -
Código del sistema difuso	- 144 -

Índice de figuras

Figura 1.- Robots bípedos y humanoides desarrollados por HONDA.	- 13 -
Figura 2.- Paradigmas de inteligencia artificial desde el punto de vista del cómputo suave.	- 15 -
Figura 3.- PNP propuesta para pasar una pelota entre dos robot. (12).....	- 19 -
Figura 4.- Red de Petri propuesta para evitar colisiones entre robots. (8).....	- 20 -
Figura 5.- Resultados del funcionamiento del controlador propuesto en (8).....	- 20 -
Figura 6.- Diagrama de funcionamiento de la solución propuesta.	- 23 -
Figura 7.- Simbología usada en las redes de Petri.....	- 26 -
Figura 8.- Ejemplo de una red de Petri marcada.....	- 27 -
Figura 9.- Ejemplos de rdP cuando se dispara una transición	- 27 -
Figura 10.- Ejemplo de una rdP <i>segura</i>	- 29 -
Figura 11.- Ejemplos de una rdP (a) Viva y (b) Cuasi-viva.	- 30 -
Figura 12.- Ejemplos (a) una rdP con un punto muerto y (b) una rdP libre de puntos muertos. .	- 31 -
Figura 13.- Ejemplo de un conflicto en una red de Petri.....	- 32 -
Figura 14.- Ejemplos de invariantes en rdP.....	- 33 -
Figura 15.- Ejemplo de una red de Petri de la cual se calculan sus propiedades.....	- 34 -

Figura 16.- Ejemplo de distintas secuencias de transiciones para una rdP.	- 37 -
Figura 17.- Representación de un sistema difuso simple.	- 40 -
Figura 18.- Función de membresía que indica el grado de pertenencia.....	- 41 -
Figura 19.- Partes de una función de membresía.	- 42 -
Figura 20.- Conjuntos difusos que representan las velocidad de giro en rpm de un motor.	- 44 -
Figura 21.- Unión de dos conjuntos difusos que representa la salida de un sistema.....	- 47 -
Figura 22.- Defusificación por método de altura.	- 47 -
Figura 23.- Defusificación por método del centroide.	- 48 -
Figura 24.- Defusificación por método del promedio ponderado.	- 48 -
Figura 25.- Defusificación por máximo medio.	- 49 -
Figura 26.- Modelo de un sistema difuso de inferencia (FIS).....	- 50 -
Figura 27.- Ejemplo de un sistema difuso de inferencia Mamdani.....	- 53 -
Figura 28.- Contenido del Bioloid kit Premium.	- 55 -
Figura 29.- Configuraciones tipo humanoide del robot bioloid.	- 55 -
Figura 30.- Unidad central de procesamiento del robot bioloid (CM-510).....	- 56 -
Figura 31.- Ejemplo de la pantalla del programa RoboPlus Task.	- 57 -
Figura 32.- Ejemplo del programa RoboPlus Motion.	- 58 -
Figura 33.- Ejemplo del programa RoboPlus Manager.	- 58 -
Figura 34.- Módulo emisor y receptor Zigbee ZIG-100/110-A.....	- 59 -
Figura 35.- Diagrama del funcionamiento de la comunicación Zigbee.....	- 60 -
Figura 36.- Adaptador para comunicación Zigbee con la computadora.....	- 60 -
Figura 37.- Motor Dynamixel AX-12.....	- 61 -
Figura 38.- Cámara HAVIMO 2.0.	- 62 -
Figura 39.- Interfaz de calibración HAVIMO-GUI	- 63 -
Figura 40.- Ejemplo del algoritmo de detección de color de la cámara HAVIMO.....	- 64 -
Figura 41.- Sensor infrarrojo.	- 65 -
Figura 42.- Curva de operación del sensor infrarrojo.	- 65 -
Figura 43.- Diagrama de la posición de los motores en el robot Bioloid.	- 66 -
Figura 44.- Robot Bioloid.....	- 67 -
Figura 45.- Montaje de la cámara.	- 67 -
Figura 46.- Giro del motor en el cual se montó la cámara.....	- 68 -
Figura 47.- Movimiento de paneo de la cámara.	- 68 -
Figura 48.- Montaje de los sensores infrarrojos.	- 69 -
Figura 49.-Operación de los sensores infrarrojos.	- 69 -
Figura 50.- Gráfica de las mediciones del sensor frontal (medición del sensor Vs. distancia).	- 72 -
Figura 51.- Ajuste de curva del sensor frontal.	- 72 -
Figura 52.- Gráfica de las mediciones de los sensores laterales (medición del sensor Vs. distancia). -	73 -
Figura 53.- Ajuste de curva de los sensores laterales.	- 73 -
Figura 54.- Posiciones en las cuales se realizó la calibración.	- 74 -
Figura 55.- Imágenes que muestran la calibración.	- 75 -
Figura 56.- Funcionamiento del algoritmo de detección de color de la cámara HAVIMO.	- 75 -

Figura 57.- Gráfica de número de píxeles Vs. distancia al objetivo.	- 77 -
Figura 58.- Ajuste de curva para calcular la distancia mediante la cámara HAVIMO.	- 77 -
Figura 59.- Movimientos proporcionados por el fabricante que puede realizar el robot.	- 78 -
Figura 60.- Postura del robot al caminar.....	- 79 -
Figura 61.- Movimiento para derribar el objetivo.....	- 80 -
Figura 62.- Zona donde debe encontrarse el centro del objetivo.....	- 82 -
Figura 63.- Diagrama del funcionamiento del sistema difuso de inferencia.	- 83 -
Figura 64.- Conjuntos difusos de los sensores laterales.	- 84 -
Figura 65.- Conjuntos difusos del sensor frontal.	- 85 -
Figura 66.- Conjuntos difusos finales del sensor frontal.....	- 86 -
Figura 67.- Conjuntos difusos de salida.....	- 87 -
Figura 68.- Red de Petri usada en la navegación del robot humanoide.	- 93 -
Figura 69.- Lugares y transiciones de la red de Petri.	- 94 -
Figura 70.- Red de Petri con marcaje m1	- 97 -
Figura 71.- Interfaz desarrollada en C++ para controlar al robot bioloid.	- 104 -
Figura 72.- Ejemplo 1 de la interfaz en operación cuando el objetivo ha sido encontrado.	- 105 -
Figura 73.- Ejemplo 2 de la interfaz en operación cuando el objeto ha sido encontrado.	- 105 -
Figura 74.- Ejemplo de la interfaz en operación cuando el robot está explorando el entorno. .	- 106 -
Figura 75.- Aplicación VOCE para el reconocimiento de ordenes por voz.....	- 107 -
Figura 76.- Materiales con los que se habilitó el área de trabajo.....	- 109 -
Figura 77.- Objetivo del robot.....	- 109 -
Figura 78.- Área de trabajo de las configuraciones 1 y 2.	- 109 -
Figura 79.- Área de trabajo configuración 1.....	- 110 -
Figura 80.- Ejemplo del funcionamiento del robot en la configuración 1.	- 110 -
Figura 81.- Otro ejemplo del funcionamiento del robot en la configuración 1.	- 111 -
Figura 82.- Diagrama configuración 2.	- 111 -
Figura 83.-Ejemplo 1 del funcionamiento del robot en la configuración 2.	- 112 -
Figura 84.- Ejemplo 2 del funcionamiento del robot en la configuración 2.....	- 113 -
Figura 85.- Área de trabajo para las configuraciones 3 y 4.....	- 113 -
Figura 86.- Diagrama configuración 3.	- 114 -
Figura 87.- Configuración 3.	- 114 -
Figura 88.- Ejemplo 1 del funcionamiento del robot en la configuración 3.....	- 115 -
Figura 89.- Ejemplo 2 del funcionamiento del robot en la configuración 3.....	- 116 -
Figura 90.- Diagrama configuración 4.	- 117 -
Figura 91.- Ejemplo 1 del funcionamiento del robot en la configuración 4.....	- 117 -
Figura 92.- Ejemplo 2 del funcionamiento del robot en la configuración 4.....	- 118 -

Índice de Tablas

Tabla 1.- Especificaciones del actuador AX-12.....	- 61 -
Tabla 2.- Valores de distancia medidos del sensor frontal.	- 71 -
Tabla 3.- Valores de distancia medidos de los sensores laterales.	- 71 -

Tabla 4.- Medida del número de píxeles del objetivo a distintas distancias	- 76 -
Tabla 5.- Posiciones del movimiento de paneo.	- 80 -
Tabla 6.- Coeficientes de las funciones de membresía de los sensores laterales.....	- 84 -
Tabla 7.- Coeficientes de las funciones de membresía del sensor frontal.....	- 85 -
Tabla 8.- Coeficientes finales de las funciones de membresía del sensor frontal.	- 86 -
Tabla 9.- Coeficientes de las funciones de membresía de salida.....	- 87 -
Tabla 10.- Reglas del sistema difuso.	- 91 -

Capítulo 1

Introducción

Este capítulo está dedicado a presentar una idea general sobre el trabajo con robots humanoides, además de hablar sobre algunos enfoques que se han usado para solucionar el problema de la navegación y mediante el estado del arte establecer el contexto en el cual fue desarrollado este trabajo, así como definir la propuesta para resolver el problema de navegación y los objetivos que deben alcanzarse.

1.1.-Los robots humanoides hoy en día (1)

Muchos aspectos de la vida moderna involucran el uso de máquinas inteligentes capaces de funcionar e interactuar de manera dinámica con su entorno. Observando esto, el movimiento de bípedos es de especial interés al referirnos a robots similares a humanos. Los robots humanoides vistos como una máquina antropomórfica caminante han estado presentes por más de veinte años. Actualmente, la investigación en robots humanoides y locomoción de bípedos es uno de los temas más emocionantes en el campo de la robótica. Hay más de 50 importantes proyectos con robots humanoides alrededor del mundo, además de muchos otros proyectos con bípedos (una lista de estos proyectos puede consultarse en (2)). La razón del creciente interés en esta rama de investigación es que las principales áreas se han vuelto evidentes. Los robots humanoides se espera que sean sirvientes y máquinas de mantenimiento, con el principal objetivo de ayudar en las tareas de la vida diaria de las personas y reemplazar a los humanos en tareas peligrosas. Es obvio, así como interesante, que los robots antropomórficos bípedos sean potencialmente capaces de moverse eficientemente en entornos no estructurados donde lo hacen los humanos. Por lo tanto, particularmente, el campo de robots de servicio, aplicaciones medicas, y tareas en entornos peligrosos son de principal importancia. Otra importante razón de la creciente investigación en robots humanoides es debido al desarrollo de tecnología en el diseño y la producción de sensores, actuadores y unidades de cómputo.

Recientemente, se ha hecho un progreso significativo en el diseño de una plataforma para robots humanoides y su control, particularmente en el desarrollo de un caminado dinámico para varios de ellos. Por ejemplo, en 1986, HONDA comenzó la investigación y un programa de desarrollo de humanoides, que resultó en una serie de prototipos (P1,P2,P3 and ASIMO) (3) . Los elementos clave del robot humanoide desarrollado por HONDA incluyen "inteligencia y movilidad" con la intención de usarlo en la vida diaria, en vez de un robot construido con el propósito de realizar tareas específicas. El diseño incluye una tecnología de movilidad de dos piernas para hacerlo compatible con la mayor cantidad de terrenos, incluyendo superficies muy rugosas.



Figura 1.- Robots bípedos y humanoides desarrollados por HONDA.

Los problemas de estabilidad en el caminado de un robot humanoide son una parte crucial en el proceso de control. Una manera de ver el caminado de los robots humanoides es clasificándolos en tres diferentes categorías (4). La primera categoría representa a los robots estáticos, cuyo movimiento es muy lento por lo tanto la estabilidad del sistema es completamente descrita por la proyección normal del centro de gravedad, que sólo depende de la posición de las articulaciones. La segunda categoría representa a los robots dinámicos, robots bípedos con pies y tobillos articulados. La estabilidad de postura de un caminante dinámico depende de la posición pero también de la velocidad y la aceleración de las articulaciones. Estos robots son potencialmente capaces de moverse de manera estática si cuentan con piernas lo suficientemente largas pero su movimiento es lento. La tercera categoría representa a los robots puramente dinámicos, los cuales no tienen pies.

Para todas las categorías antes mencionadas de robots caminantes, el problema de estabilidad y confiabilidad del caminado bípedo es el problema principal y se mantiene aún sin resolver con un alto grado de fiabilidad.

Las aplicaciones con robots humanoides usualmente requieren que el robot sea inteligente. Los robots humanoides inteligentes son dispositivos orientados a la funcionalidad, construidos para realizar un conjunto de tareas. Son sistemas autónomos capaces de extraer información de su entorno, usar su conocimiento sobre el mundo, la inteligencia de sus funciones y sus capacidades para realizar una tarea. Los robots humanoides inteligentes deben ser autónomos para moverse de manera segura y significativa con un propósito, es decir, deben aceptar tareas donde el usuario especifique qué es lo que se quiere hacer en vez de cómo se debe hacer, y que además las lleven a cabo sin mayor intervención humana. Deben de ser inteligentes para determinar todas las posibles acciones en un entorno dinámico impredecible usando la información de varios sensores. Los operadores de los robots pueden transferir su conocimiento, experiencia y habilidades, para que sean capaces de completar tareas complejas.

Naturalmente, la primera aproximación para hacer a los robots humanoides más inteligentes fue la integración de sofisticados sistemas de sensores como la visión por computadora, sensores táctiles, sonares y sensores ultrasónicos, escáneres laser y otros sensores inteligentes. Sin embargo, hoy en día los sensores aún están muy limitados en interactividad y en adaptabilidad a entornos cambiantes.

Por otra parte, para diseñar robots y sistemas que mejor se adapten a su entorno, es necesaria incluir investigación en el campo del diseño mecánico de robots, sistemas de percepción del entorno y control inteligente embebido que haga frente con la complejidad de la tarea, toma de decisión multi-objetivo, percepción de grandes volúmenes de información y una cantidad sustancial de información heurística. También en el caso de que el robot se desplace en un entorno desconocido, el conocimiento tal vez no sea suficiente. Por lo tanto, el robot tiene que adaptarse al entorno y ser capaz de adquirir nuevo conocimiento mediante un proceso de aprendizaje. El aprendizaje de robots está especialmente interesado en equipar a los robots con la capacidad de mejorar su comportamiento conforme pasa el tiempo, basado en sus experiencias adquiridas.

Hay distintos paradigmas de inteligencia que son capaces de resolver problemas de control inteligente en robots humanoides. La teoría conexionista (RNA - redes neuronales artificiales), la lógica difusa (LD) y la teoría de computación evolutiva (AG - algoritmos genéticos) son de gran importancia en el desarrollo de algoritmos para el control inteligente de robots humanoides. Debido a su alto aprendizaje y habilidades cognitivas, su buena tolerancia a la incertidumbre e imprecisión, las técnicas inteligentes han

encontrado un amplio campo de aplicación en el área del control de robots. También son de gran importancia en el desarrollo de algoritmos eficientes las técnicas híbridas basadas en la integración de técnicas particulares como son las redes neuro-difusas, los algoritmos neuro-genéticos y los algoritmos difuso-genéticos.

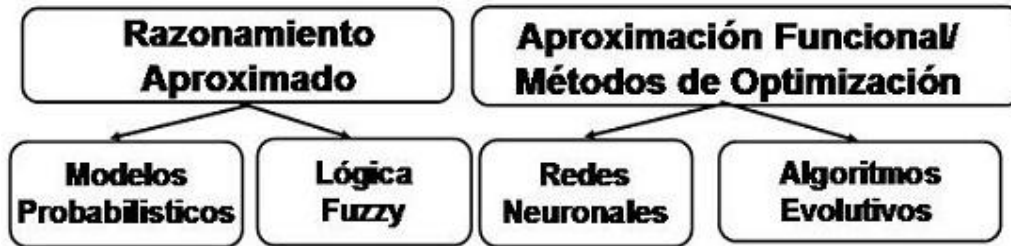


Figura 2.- Paradigmas de inteligencia artificial desde el punto de vista del computo suave.

Los sistemas de control difuso (5) (6) basados en la formulación matemática de lógica difusa, tienen la capacidad de representar el conocimiento humano y las experiencias como un conjunto de reglas difusas. Los controladores difusos para robots usan el conocimiento humano de cómo hacer las cosas o reglas heurísticas en forma lingüística **SI-ENTONCES**, mientras que el sistema de inferencia difuso calcula la acción de control eficiente para un determinado propósito.

1.2.-La navegación en robots móviles.

La navegación es un aspecto importante en el campo de robots móviles. La navegación es encontrar un camino adecuado libre de colisiones para que el robot móvil se mueva desde una configuración inicial hasta un objetivo dentro de un ambiente compuesto por obstáculos. En relación a esto es muy frecuente que se busque que este camino sea óptimo o cercano al óptimo con respecto al tiempo, distancia, energía o uniformidad del desplazamiento. La distancia es un criterio comúnmente adoptado. Desde las últimas décadas, la navegación de robots ha sido una área emergente, y muchas técnicas han sido tomadas para atacar este problema, algunas son las técnicas de lógica difusa, algoritmos genéticos, métodos de campo potencial, acercamientos con redes neuronales e incluso técnicas híbridas. Cada método tiene sus propias ventajas en relación a los otros en algunos aspectos. Generalmente, las mayores dificultades del problema de navegación de robots son la complejidad computacional, los mínimos locales y la adaptabilidad. Los

investigadores se mantienen constantemente buscando alternativas y maneras más eficientes de resolver este problema. (7)

La navegación y control de robots móviles ha sido vista desde distintos enfoques, uno es en el cual, el robot no adquiere información del entorno en tiempo real, sino que por medio de información a priori relacionada al medio ambiente donde se desplaza el robot es capaz de encontrar una ruta al enviar esa información a un algoritmo y luego a un planeador de caminos el cual creará una ruta desde la posición inicial del robot hasta su objetivo. Por otra parte, existen algoritmos de navegación de robots que tratan de encontrar una ruta en tiempo real. Recientemente estos algoritmos se han vuelto de mayor interés para los investigadores de esta área, aunque, también tienen algunas limitantes y restricciones. Por ejemplo, es difícil desplazar a un robot en un entorno lleno de obstáculos que no pueda ser descrito como un modelo matemático; para después pasar esta información a un algoritmo de navegación de robots. Así que, inspirado de los humanos, un acercamiento desde la lógica difusa ha sido recientemente aplicado a muchos algoritmos de navegación para robots móviles. (8)

La lógica difusa tiene una característica importante en la manera de tratar con varias situaciones sin necesitar de un modelo analítico del entorno. Por esta razón, muchos investigadores abordan este problema proponiendo algoritmos de navegación por medio del uso de la lógica difusa. Sin embargo, en casos en los cuales es necesario que los robots móviles se desplacen en ambientes complejos, estos métodos tienen desventajas al tener que construir las reglas de navegación de manera consistente, lo cual en algunas ocasiones resulta bastante complicado. Debido a esto, para superar esta nueva problemática otros métodos han sido propuestos, como los algoritmos basados en el comportamiento, o bien el uso de otras técnicas, como es el caso de este trabajo donde se propone el uso de las redes de Petri.

1.3.-Las redes de Petri (9)

Las redes de Petri son una herramienta gráfica y matemática de modelado aplicable a muchos tipos de sistemas. Son una herramienta prometedora para describir y estudiar el procesamiento de la información en sistemas que están caracterizados por ser concurrentes, asíncronos, distribuidos, paralelos, no deterministas y/o estocásticos. Como una herramienta gráfica, las redes de Petri pueden ser usadas como una ayuda de comunicación visual similar a los diagramas de flujo, los diagramas de bloques y las redes. Además, las marcas (tokens) son usadas en estas redes para simular las actividades

dinámicas y concurrentes del sistema. Como una herramienta matemática, es posible definir una ecuación de estado, ecuaciones algebraicas y otros modelos matemáticos que gobiernan el comportamiento del sistema. Las redes de Petri pueden ser usadas tanto por teóricos como por usuarios no especializados que desean utilizar las bondades de esta herramienta de modelado.

Las redes de Petri han sido propuestas para una gran variedad de aplicaciones. Esto es debido a la generalidad y la tolerancia incluidas en las redes de Petri. Pueden aplicarse de manera informal en muchas áreas o sistemas que puedan ser descritos de manera gráfica, como diagramas de flujo y que además necesiten una manera de representar actividades paralelas o concurrentes. No obstante, debe ponerse mucho cuidado en encontrar el punto medio entre la generalidad del modelo y las capacidades de análisis. Esto es, entre más general sea el modelo, se vuelve menos accesible para el análisis. De hecho, una de las mayores desventajas de las redes de Petri es el problema de la complejidad, esto es, que los modelos basados en redes de Petri tienden a volverse muy grandes para el análisis aún de un sistema de tamaño moderado. En la aplicación de las redes de Petri, es frecuentemente necesario agregar modificaciones o restricciones especiales que encajen con una aplicación específica.

Ahora se presentan algunas investigaciones que están relacionadas con este trabajo que sirven para entender el contexto en el cual fue realizado o bien como un punto de partida de como se han llevado a cabo estudios similares en otros países y cuáles han sido los resultados, limitantes, logros o aportaciones al integrar la lógica difusa y las redes de Petri para realizar tareas con robots.

1.4.- Estado del Arte

En *Petri net models of robotic tasks* (10), publicado en el 2002, se plantea el uso de las redes de Petri para modelar las actividades de un robot, Este artículo utiliza las propiedades de las rdP para hacer un análisis cualitativo (usando redes no temporizados) y cuantitativo (usando redes temporizadas o estocásticas) de las actividades del robot. La idea principal del trabajo recae en expresar las tareas del robot como una cadena de *tareas primarias*, que representan la secuencia de acciones que debe llevar a cabo el robot para completar el objetivo de la tarea. Donde cada *tarea primitiva* puede ser implementada a base de más de una *acción primaria*, por ejemplo, se considera como una *tarea primaria* localizar un objeto, la cual puede realizarse mediante el uso de distintos algoritmos de procesamiento de imágenes, que son considerados *acciones primarias*. La idea de describir las tareas y las

acciones del robot mediante el uso de redes de Petri es usada en el diseño de la rdP para la navegación del robot humanoide contenida en esta tesis.

En relación al uso de los sistemas difusos para la navegación espacial, un buen ejemplo se puede ver en *Fuzzy logic based decision making system for collision avoidance of ocean navigation under critical collision condition* (11), publicado en el 2010, donde se propone el uso de un sistema difuso de inferencia para evitar colisiones entre barcos en el océano. Los resultados de este trabajo son analizados mediante una simulación hecha en MATLAB, los cuales son satisfactorios incluso para condiciones consideradas críticas. Aquí se puede vislumbrar la versatilidad de los sistemas difusos de inferencia que pueden ser empleados para muy distintos objetivos, pero cuyo potencial recae en la definición de las reglas de inferencia basadas en el conocimiento del problema y que precisamente es el punto fuerte de (11) ya que emplea reglas para evitar colisiones en el océano aceptadas internacionalmente con el objetivo de obtener los mejores resultados del FIS para generar una alternativa de navegación autónoma.

Un artículo que resulta interesante debido a que brinda un "framework" para usar las redes de Petri aplicadas a sistemas con múltiples robots es *Petri net plans - A framework for collaboration and coordination in multi-robot systems* (12), que fue publicado en el 2010, en el se presentan las "Petri net plans", abreviado como PNP, el cual es un lenguaje basado en las redes de Petri ordinarias, pero mediante el cual es posible diseñar de manera intuitiva y efectiva el comportamiento de un robot o múltiples robots, además de brindar características esenciales en el desarrollo de aplicaciones con robots. Estas características son el procesar información de los sensores, de las interrupciones y de la concurrencia. También las PNP permiten un análisis formal basado en las propiedades de las rdP ordinarias. En (12) primero se hace una definición formal de las PNP y después de la misma manera se demuestran y se describen algunas de sus propiedades. Después se reportan tres casos de estudio en los cuales han sido utilizadas las PNP, donde las plataformas empleadas son: un robot con ruedas usado para misiones de búsqueda y rescate y un robot cuadrúpedo AIBO usado para jugar futbol soccer. Los casos de estudio fueron los siguientes: El primero es el uso de un robot para la exploración y búsqueda en un ambiente desconocido, colaboración de múltiples robots para la recolección y el último es pasar una pelota entre robots que juegan futbol soccer. Los resultados de estos casos de estudio pueden ser conocidos con mayor detalle en (13) para el caso uno, (14) para el caso dos. En la figura, se muestran la PNP utilizada para el tercer caso de estudio.

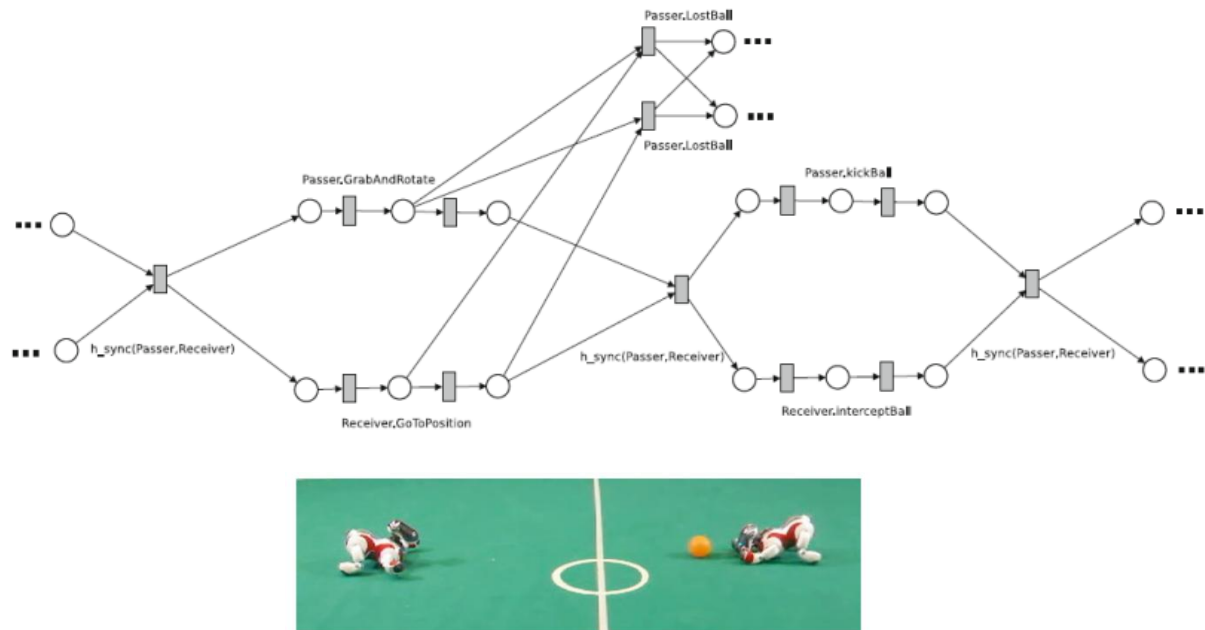


Figura 3.- PNP propuesta para pasar una pelota entre dos robot. (12)

En *Navigational control of several mobile robotic agents using Petri-potential-fuzzy hybrid controller* (8), publicado en el 2011, se propone el uso de un controlador difuso potencial, es decir un sistema híbrido entre un sistema difuso y un método de campo potencial, junto con una red de Petri para la navegación de varios robots. En el cual la plataforma usada es un robot móvil con ruedas. El controlador difuso potencial es el encargado de la navegación y de evitar los obstáculos para que pueda desplazarse a la posición objetivo, para lograr esto, el robot móvil cuenta con una arreglo de seis sensores ultrasónicos y de cuatro sensores infrarrojos mediante los cuales el robot móvil adquiere información de su entorno, en este caso, la distancia a la cual se encuentran los obstáculos. Las mediciones de los sensores son las entradas del controlador y la salida es la velocidad de las rueda izquierda y derecha del robot, entonces por medio de los cambios en las velocidades de las ruedas es como se logra cambiar la trayectoria del robot para que evite los obstáculos. En el trabajo no se especifica el número de reglas que fueron propuestas para el sistema difuso, pero se muestran 16 reglas y se menciona que sólo son algunas de las que usa el controlador híbrido. La red de Petri se utiliza solamente para resolver conflictos entre robots, es decir, decidir cual tiene prioridad de avanzar en caso de que uno este interfiriendo en el camino del otro, a este proceso lo llama negociación ya que debe decidir cuál de los dos robot se queda quieto y funge como un obstáculo, mientras que el otro sigue avanzando, la red propuesta en (8) es bastante simple y no se le hace ningún análisis de propiedades.

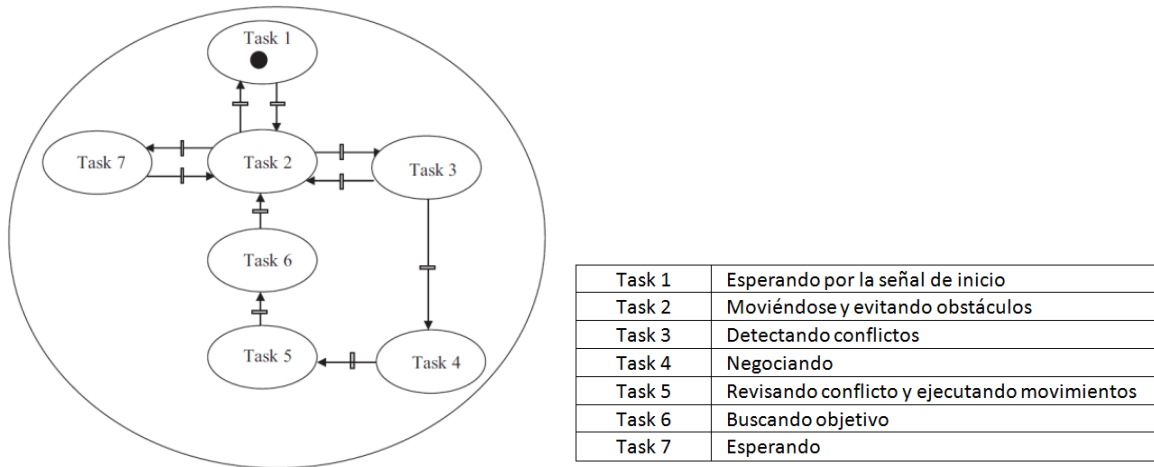


Figura 4.- Red de Petri propuesta para evitar colisiones entre robots. (8)

Los resultados de (8) se muestran mediante un programa que realiza una simulación y también en experimentos prácticos. En las simulaciones se prueba el controlador para múltiples robots, y el resultado es que cada robot es capaz de alcanzar el objetivo más cercano a su posición. Los experimentos prácticos se realizan usando un robot móvil con ruedas en un ambiente restringido, donde se colocan los obstáculos en distintas configuraciones para probar el desempeño del controlador, en este caso se reportan 3 configuraciones distintas, la primera configuración y el funcionamiento del robot puede verse en la siguiente figura.

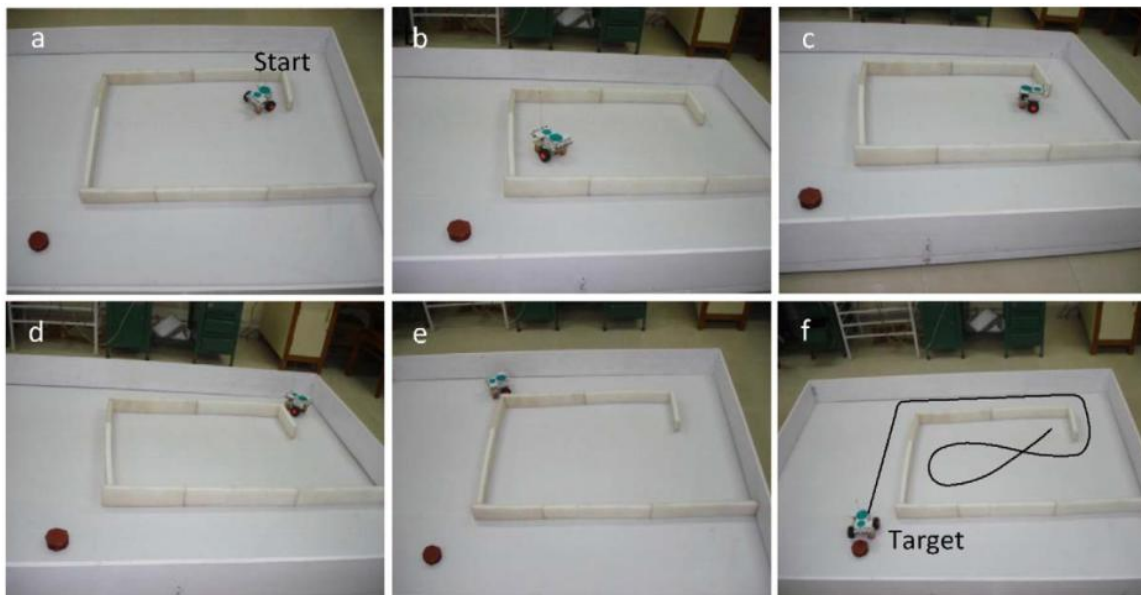


Figura 5.- Resultados del funcionamiento del controlador propuesto en (8).

1.5.-Justificación

Después de conocer un poco acerca del trabajo que se ha realizado en relación a la navegación en robots móviles en los últimos años, se presentaron los aspectos importantes que deben tomarse en cuenta y los problemas presentes en este tipo de trabajos. Lo anterior nos brinda un contexto para comenzar a hablar del objetivo de este trabajo, el cual se enfoca en la navegación de un robot humanoide con el propósito de desplazarse en un entorno restringido para alcanzar un objetivo específico (llegar a cierto punto del entorno) y realizar una tarea sencilla (en este caso, derribar un objeto).

Debido a la problemática constante de la navegación de robots, es que muchos trabajos han sido propuestos para tratar de resolver esta cuestión, algunos solo empleando técnicas de cómputo suave, técnicas de control convencional, analíticas o híbridas. Debido a que no existe una sola alternativa para resolver esta problemática, ya que todas tienen sus ventajas y desventajas, es que sigue siendo un área de gran interés para los investigadores.

Por esta razón es que surge la idea de este trabajo al proponer el uso de un sistema difuso, el cual ha sido usado en distintas aplicaciones y que ha demostrado tener un buen funcionamiento, y agregarle otra herramienta que pueda complementarlo, ya que las técnicas híbridas recientemente se han vuelto muy utilizadas ya que conjugan las ventajas de dos enfoques distintos. En este caso se propuso utilizar las redes de Petri de manera conjunta a la lógica difusa. La idea fue tratar de encontrar una alternativa simple para resolver el problema de la navegación de un robot humanoide, ya que muchos de los enfoques que pueden encontrarse en la literatura utilizan sistemas complejos basados en ecuaciones diferenciales y en conceptos, que para lograr comprenderlos a fondo es necesario contar con una base previa de distintos conocimientos. Sin embargo al proponer que la solución fuera simple no implica que sea una solución mala o pobre, ya que se buscó que fuera una solución robusta, y que permita realizar la tarea de manera correcta, pero que el concepto general fuese simple de comprender.

Se decidió utilizar la lógica difusa, porque es muy similar a la lógica que utilizamos en la vida diaria para procesar conceptos que no están estrictamente acotados, lo que la vuelve relativamente sencilla de comprender, y que utiliza reglas lingüísticas que son sencillas de proponer en base a la experiencia con la que se cuenta en relación al problema que se busca resolver. Mientras que las redes de Petri son una herramienta gráfica que describe tareas o eventos de manera simple e intuitiva, pero que guarda en su definición formal una gran cantidad de información referente al sistema o al problema que describen.

1.6.-Solución propuesta

La estrategia que se propuso para resolver el problema antes mencionado fue el utilizar un sistema difuso que se encargue de la navegación del robot humanoide en el entorno restringido, que le diga cuando girar y hacia qué lado dependiendo de la información recibida por los sensores con los que cuenta el robot. De manera conjunta al sistema difuso se utilizó una red de Petri que describe y administra las acciones del robot, con la finalidad de ser un supervisor y controlar la ejecución de algunas de las tareas del robot o de activar el sistema difuso para la exploración, logrando de esta forma poder alcanzar el objetivo buscado.

La plataforma empleada para desarrollar este trabajo fue un robot humanoide BIOLOID-PREMIUM al cual se le montaron 3 sensores infrarrojos que miden distancia (DMS por sus siglas en inglés) con ellos el robot tiene una percepción de su entorno, aunque limitada, que es la base para que pueda moverse dentro del entorno restringido. El robot se comunica a una computadora personal mediante el protocolo Zigbee, el cual consiste en transmisión de datos por radiofrecuencia, esto es necesario porque debido a las limitadas capacidades del BIOLOID para procesar información, no es posible implementar el sistema difuso en él. Sin embargo, el que dependa de la comunicación con una computadora no afecta la autonomía del robot humanoide ya que una vez que se le indica que comience la tarea, no requiere de otra intervención humana hasta que alcance su objetivo.

Un elemento esencial para que el robot humanoide sea capaz de encontrar y dirigirse al objetivo que está siendo buscado es la cámara. Al robot se le montó una cámara HAVIMO 2.0 la cual cuenta con un procesamiento de imágenes que es realizado en un circuito integrado con el objetivo de aislar regiones de color y extraer información de estas, como esta cámara es compatible con la unidad de procesamiento del robot BIOLOID es posible recibir esta información y trabajar con ella dentro del procesador del robot, sin embargo no es posible hacer otro tratamiento a las imágenes, ya que no se tiene acceso a los píxeles, o a la matriz que representa la imagen captada por la cámara. Por esta razón es mejor considerar a la cámara HAVIMO 2.0 como un sensor de color en vez de como una cámara digital convencional.

Para probar el desempeño de lo antes propuesto se habilitó un área de trabajo restringida, ya que para que el robot pueda desplazarse de la mejor manera posible es necesario que sea sobre una superficie rígida y lisa, también es necesario reducir el número de colores que pueda percibir la cámara HAVIMO 2.0, las cuestiones referentes a esto, se tratarán con mayor detalle más adelante en este trabajo.

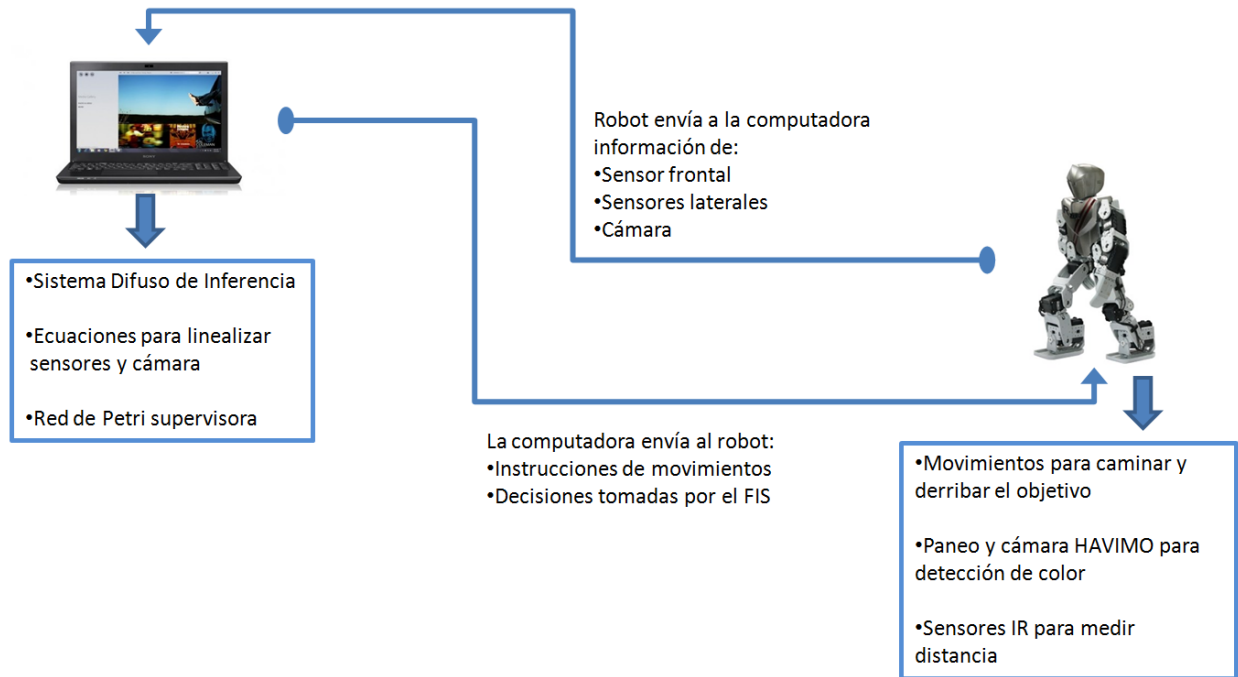


Figura 6.- Diagrama de funcionamiento de la solución propuesta.

1.7.-Objetivos

1.7.1.-Objetivo general

Implementar un sistema difuso de toma de decisión (FDMS), también conocido como un sistema difuso de inferencia (FIS), para determinar el giro que realizará un robot Bioloid al desplazarse dentro de un ambiente controlado, mientras busca un objeto; así como utilizar las redes de Petri (rdP) para modelar las actividades del robot, con el objetivo de administrar tareas recursos y acciones además de supervisar el funcionamiento del robot.

1.7.2.-Objetivos específicos

1. Uso de la cámara HAVIMO 2.0 para la detección de objetos mediante su color.
2. Controlar los movimientos del robot mediante el software proporcionado por el fabricante.
3. Implementar un sistema difuso de toma de decisión (FDMS) para determinar la dirección giro o el número de pasos que realizará el robot.
4. Modelar mediante redes de Petri las actividades y posibles conflictos en las acciones del robot, y analizar las propiedades de la RdP propuesta.

Capítulo 2

Marco Teórico

En este capítulo se define de manera formal y analítica las herramientas que se propusieron para resolver el problema de la navegación de un robot humanoide en un ambiente controlado mediante el uso de redes de Petri y lógica difusa. Se explican teóricamente los conceptos y las definiciones con los cuales se llevó a cabo el desarrollo y en base a los cuales se propuso el modelo híbrido de la solución. En el caso de las redes de Petri, se define su composición, sus propiedades y cuál debe ser el proceso para encontrarlas propiedades de las rdP. Con respecto a la lógica difusa, se explica cual debe ser el proceso para generar un sistema difuso de inferencia.

2.1.-Redes de Petri (9) (15)

El concepto de las redes de Petri tiene su origen en la disertación de Carl Adam Petri, presentada en 1962 a la facultad de matemáticas y física de la universidad técnica de Darmstadt, Alemania Occidental.

Los usuarios principales de las redes de Petri son científicos en control automático o en computación. Sin embargo, esta herramienta es suficientemente general para modelar fenómenos extremadamente distintos. Las redes de Petri presentan dos características muy interesantes. La primera, es que hacen posible modelar y visualizar comportamientos con paralelismo, concurrencia, sincronización y distribución de recursos. La segunda, es que los resultados teóricos concernientes a ellas son abundantes; las propiedades de estas redes han sido y aún son extensamente estudiadas.

2.1.1.-Lugares, transiciones y arcos

Una red de Petri (rdP) tiene dos tipos de nodos, llamados lugares y transiciones, véase figura 7. Un **lugar** es representado por un círculo y una **transición** por una barra (algunos autores la representan con una caja). Los lugares y las transiciones están conectados por

arcos dirigidos. El número de lugares en una red debe ser finito y no cero. El número de transiciones también debe ser finito y no cero. Un arco es dirigido y conecta ya sea un lugar con una transición o bien una transición con un lugar, a este debe asignársele un peso que en el caso de las redes de Petri ordinarias, este peso es un valor que está en el dominio de los números naturales. Por lo general, se considera que los arcos tienen un peso de valor uno. Pero es posible también que su peso sea mayor que uno, en tal caso, el arco debe etiquetarse con el valor de n .



Figura 7.- Simbología usada en las redes de Petri.

En otras palabras, una rdP es un grafo bipartito, esto es, lugares y transiciones se alternan en un camino hecho por una sucesión de arcos. Es obligatorio para cada arco tener un nodo en cada extremo. (De un nodo k , lugar o transición, a un nodo h , transición o lugar, debe haber por lo menos un arco).

2.1.2.-Marcaje

La figura 8 representa una rdP marcada. Cada lugar contiene un número entero (positivo o cero) de **tokens** o **marcas**. El número de tokens contenidos en el lugar P_i se llama $m(P_i)$ o m_i . Por ejemplo, en la figura antes mencionada, tenemos $m_1 = m_3 = 1$, $m_6 = 2$ y $m_2 = m_4 = m_5 = m_7 = 0$. El marcaje de la red, \mathbf{m} , es definido por el vector de los marcajes de cada lugar, esto es, $\mathbf{m} = (m_1, m_2, m_3, m_4, m_5, m_6, m_7)$, por ello el marcaje de la figura es $\mathbf{m} = (1, 0, 1, 0, 0, 2, 0)$. El marcaje define el estado de la rdP, o más precisamente el estado del sistema descrito por la rdP. La evolución de los estados del sistema modelado corresponde a la evolución del marcaje, la cual ocurre al disparar las transiciones.

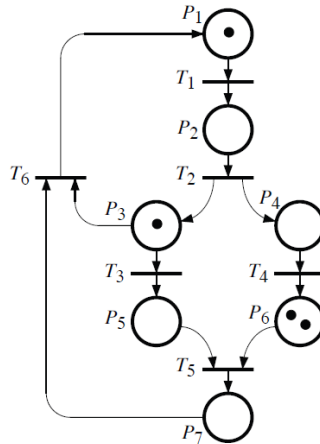


Figura 8.- Ejemplo de una red de Petri marcada.

2.1.3.-Disparar una transición

Una transición solo puede ser disparada si cada uno de sus lugares de entrada de dicha transición contienen por lo menos un token; en este caso se dice que la transición puede ser disparada o que está habilitada. En algunos casos dependiendo de los pesos de los arcos de entrada de la transición, es necesario que existan más de un token para que esta pueda habilitarse.

Disparar una transición T_j consiste en quitar un token de cada uno de los lugares de entrada de la transición T_j y en agregar un token a cada uno de los lugares de salida de la transición T_j .

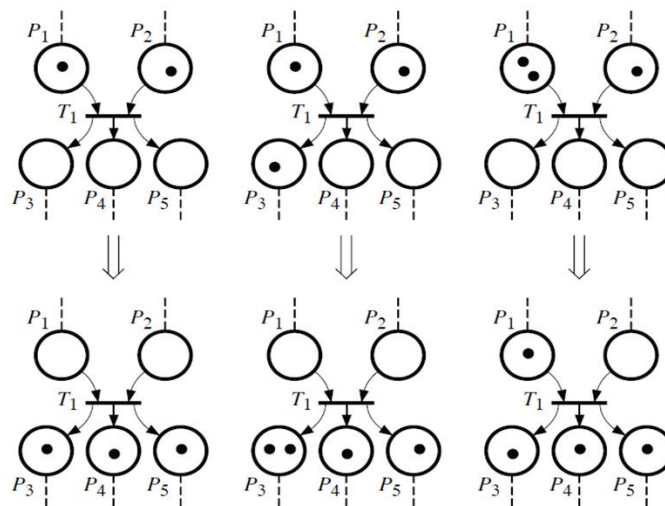


Figura 9.- Ejemplos de rdp cuando se dispara una transición

Cuando una transición está habilitada, esto no implica que esta se disparará inmediatamente. Solo se mantiene como una posibilidad. Incluso puede ser que dos transiciones estén habilitadas al mismo tiempo, aunque no se sepa cual transición se disparará, pero se sabe cuáles son los marcajes que corresponden a que se dispare cualquiera de las transiciones habilitadas.

2.1.4.-Características esenciales

La dinámica de un sistema descrito por una rdP es representado por la evolución de sus marcajes. El concepto de una red de Petri *viva* (es cuando ninguna transición deja de poder ser habilitada) y especialmente el de una red *libre de puntos muertos* son importantes. Los puntos muertos ocurren si ya no puede ser disparada ninguna transición. En casi todos los casos, esto corresponde a un sistema que está mal diseñado o bien, a uno que está mal modelado.

Una red de Petri en la cual es siempre posible regresar al marcaje inicial se dice que es *reversible*.

El concepto de conflicto frecuentemente aparece. Una estructura de conflicto existe cuando el mismo lugar es una entrada de dos o más transiciones. Si ambas transiciones están habilitadas y solo hay un token en el lugar que tienen en común, solo una de estas transiciones puede ser disparada. Por lo tanto hay un conflicto entre estas dos transiciones. Este concepto se relaciona a una decisión que debe tomarse en relación a dos posibles disparos. Lo cual puede verse, por ejemplo, en el caso en que se comparten recursos en común.

El número de tokens en una rdP puede incrementarse o decrementarse (hasta cero). Una red de Petri debe , hablando en términos generales, tener la propiedad de ser *acotada* (es decir, tener un número finito de tokens) .

Una red de Petri es *pura* si no tiene bucles, donde para un par de lugar y transición se dice que tiene un bucle si un lugar es la entrada y la salida de la transición. Se dice que una red de Petri es *ordinaria* si los pesos de todos sus arcos son uno.

Ya que conocemos algunas de las características de una rdP, resulta interesante ver las diferencias, ventajas o desventajas que existen con herramientas similares. En un grafo de estado puede tal vez haber conflictos, pero no hay sincronización (es decir, una transición con por lo menos dos lugares de entrada) . En un grafo de eventos, puede tal vez haber sincronización, pero no hay conflictos.

Cuando deseamos elegir entre un número de transiciones habilitadas es necesario agregar el concepto de prioridad a la red de Petri, para esto es necesario definir una relación de orden entre las transiciones de la red. La relación $T_j < T_k$ significa que T_j tiene mayor prioridad sobre T_k si ambas están habilitadas.

2.1.5.-Propiedades de las redes de Petri

Estas propiedades están ligadas a conceptos que denotan que una red de Petri sea acotada, viva, contenga puntos muertos y para encontrar sus componentes conservativas o sus componentes repetitivas.

2.1.5.1.-Redes de Petri acotadas y redes de Petri seguras

Se dice que un lugar P_i está acotado por un marcaje inicial m_0 , si existe un número k , entero positivo, que para todos los marcajes alcanzables desde m_0 , el número de tokens en P_i no sea mayor que k (se dice que P_i es k -acotado).

Por lo tanto una red de Petri es *acotada* para un marcaje inicial m_0 si todos sus lugares están acotados por m_0 (la rdP es k -acotada si todos sus lugares son k -acotados). Se puede decir que una red de Petri es *segura* para un marcaje inicial m_0 si y solo si para todos los marcajes alcanzables, cada lugar contiene cero o un token. Una rdP *segura* es entonces un caso particular de una rdP acotada para la cual todos los lugares son 1-acotados.

Las propiedades de una red de Petri de ser *acotada* y *segura* dependen de su marcaje inicial m_0 . Esto es claro cuando hablamos de una rdP *segura* ya que es suficiente que en m_0 uno de los lugares contenga 2 tokens para que la red no sea *segura*.

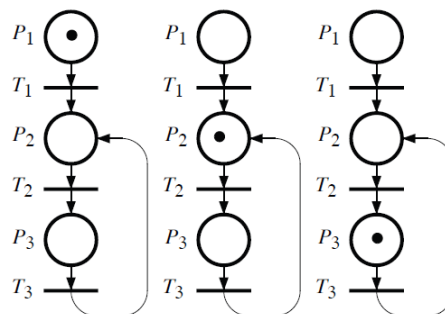


Figura 10.- Ejemplo de una rdP segura.

2.1.5.2.-Puntos muertos y redes de Petri vivas

El marcaje de una rdP cambia cuando se disparan transiciones. Cuando ciertas transiciones ya no pueden ser habilitadas para dispararse y cuando toda o una parte de la red ya no "funciona", entonces muy probablemente exista un problema en el diseño del sistema descrito. Ahora veremos los conceptos relacionados a esto.

Una transición T_j es viva para un marcaje inicial m_0 si para todos los marcajes alcanzables desde m_0 existe una secuencia de transiciones S desde m_i que contenga a la transición T_j . En otras palabras, cualquiera que sea la evolución de la red, que siempre exista la posibilidad de disparar T_j en alguna ocasión.

Una red de Petri es *viva* para un marcaje inicial m_0 si todas sus transiciones son *vivas* para m_0 . En otras palabras, si una red de Petri es viva, esto significa que sin importar la evolución de la red, ninguna de sus transiciones se inhabilitara de manera permanente.

Una transición T_j es *cuasi-viva* para un marcaje inicial m_0 , si existe una secuencia desde m_0 la cual contenga a T_j . En decir, que una transición es *cuasi-viva* si existe una posibilidad de que dicha transición pueda ser disparada por lo menos una vez.

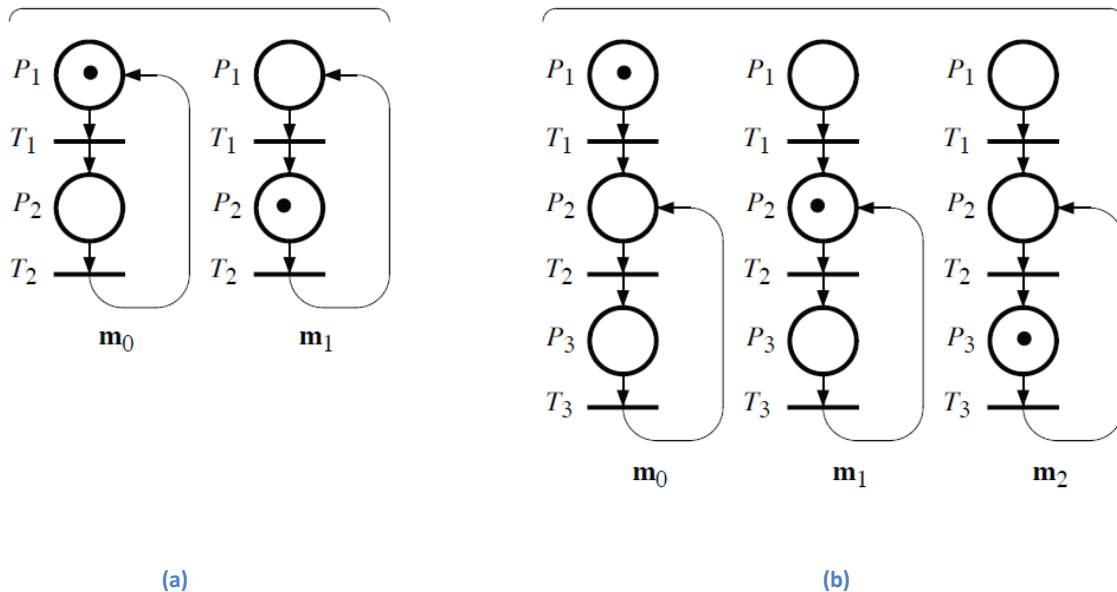


Figura 11.- Ejemplos de una rdP (a) Viva y (b) Cuasi-viva.

Un punto muerto (o sumidero) es un marcaje tal que ninguna transición este habilitada para dispararse.

En la figura 12(a) si disparamos la transición T_1 el marcaje resultante es un punto muerto, lo que significa que el marcaje ya no puede evolucionar.

Entonces una red de Petri que está *libre de puntos muertos*, es aquella que para un marcaje inicial m_0 no existan marcajes alcanzables que sean *puntos muertos*.

La figura 12(b) es una red de Petri libre de puntos muertos. Si se dispara T_1 entonces es posible disparar T_3 y luego T_5 y se mantienen esas dos últimas transiciones vivas. En cambio si se dispara T_2 en un inicio, eso habilitara a T_4 y luego a T_6 , podemos ver que se tiene un comportamiento simétrico. Con lo anterior podemos entender que una red de Petri libre de puntos muertos será aquella en la que siempre alguna de sus partes se mantenga funcionando.

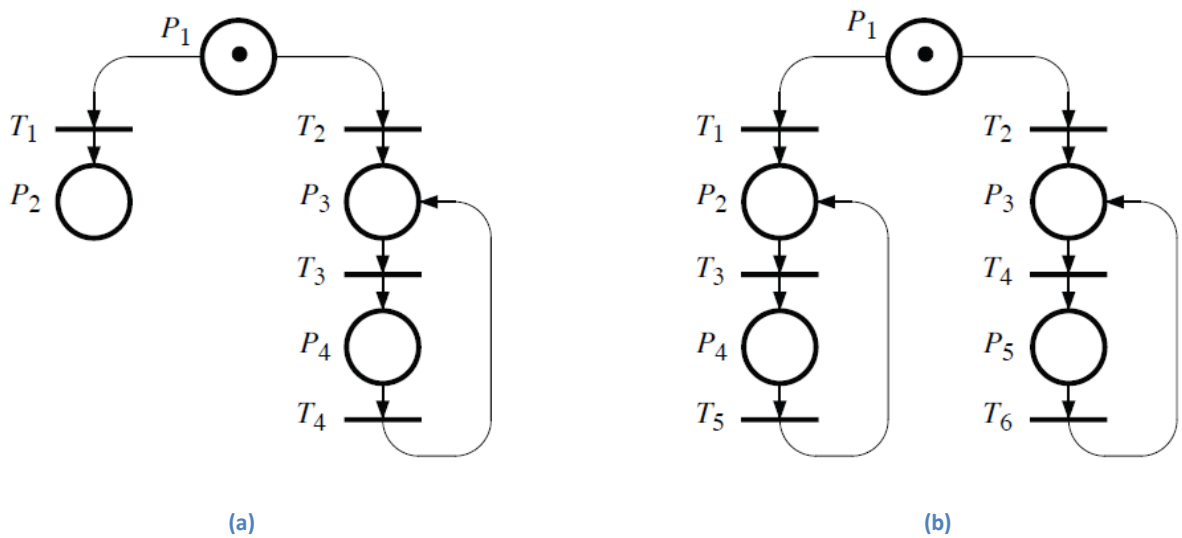


Figura 12.- Ejemplos (a) una rdP con un punto muerto y (b) una rdP libre de puntos muertos.

Una red de Petri tiene un *estado de origen* en m_h si dado un marcaje inicial m_0 existe una secuencia S_i de transiciones, tal que por medio de ellas se pueda ir de m_i , que representa cualquier marcaje alcanzable, hasta m_h .

Una red de Petri es reversible para un marcaje inicial m_0 si m_0 es un *estado de origen* (también puede referirse a esta propiedad como que es reinicializable).

Recordando que una estructura de conflicto corresponde a un conjunto de por lo menos dos transiciones T_1 y T_2 las cuales tienen como entrada un lugar en común.

Un *conflicto efectivo* es la existencia de un conflicto estructural, es decir que exista un lugar común para un conjunto de transiciones, y de un marcaje m tal que el conjunto de transiciones estén habilitadas por m y la suma de los pesos en el lugar sea menor a la suma de los pesos en los arcos que unen el lugar con las transiciones.

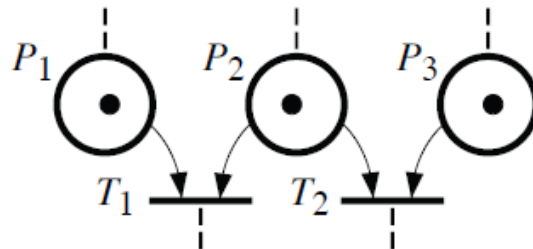


Figura 13.- Ejemplo de un conflicto en una red de Petri.

2.1.6.-Invariantes

Comenzando con un marcaje inicial, el marcaje de una red de Petri puede evolucionar al dispararse las transiciones, y, si no hay puntos muertos, el número de disparos de las transiciones es ilimitado. Sin embargo no cualquier marcaje puede alcanzarse, todos los marcajes alcanzables tiene algunas propiedades en común; a aquellas propiedades que no cambian cuando se disparan las transiciones se les llama *invariantes*. De manera similar, no cualquier secuencias de transiciones puede dispararse, algunas propiedades *invariantes* son comunes para las posibles secuencias de transiciones.

Por lo tanto los invariantes nos brindan ciertas propiedades de los marcajes alcanzables y poder entender las secuencias de transiciones, independientemente de la evolución de la red.

En la figura 14(a) podemos ver que sin importar la evolución de la red, siempre habrá un solo token para los 4 lugares. Esto es que todo el tiempo tendremos que la suma del marcaje de todos los lugares es igual a 1, $m_1 + m_2 + m_3 + m_4 = 1$. Esta propiedad *invariante* tiene un significado obvio; nos dice que todo el tiempo solo podemos tener una estación a la vez. Generalmente, esto nos dice que este componente tiene un significado físico o real en el sistema que estamos modelando, como en la figura 14(b) en la cual se mantiene un número de elementos constante.

Las secuencias de transiciones que son posibles para un marcaje m_0 de la *figura a* son los siguientes: T_1 , T_1T_2 , $T_1T_2T_3$, $T_1T_2T_3T_4$, $T_1T_2T_3T_4T_1$, etc. La secuencia $T_1T_2T_3T_4$ es de particular interés ya que nos lleva desde m_0 hasta m_0 . Lo cual nos causa regresar a un estado inicial. Por lo tanto, la secuencia tal vez quiera ser repetida. Esta es una *secuencia repetitiva*. Una *secuencia repetitiva* que contenga a todas las transiciones (por lo menos una vez) es una *secuencia repetitiva completa*.

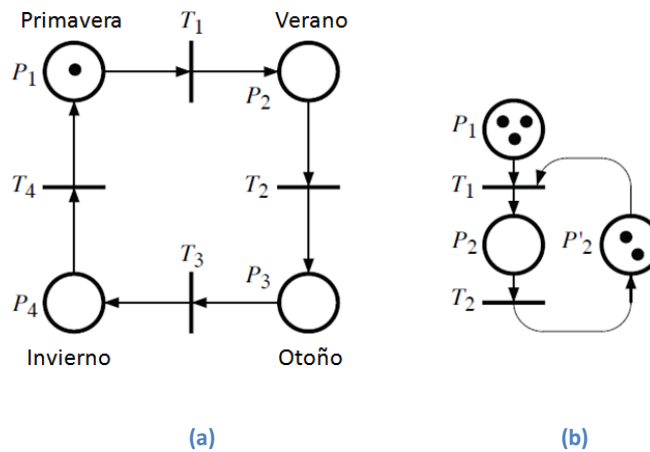


Figura 14.- Ejemplos de invariantes en rdP.

2.1.7.-Buscando las propiedades de las redes de Petri

Se presentara de manera formal la definición de las redes de Petri junto con algunas de sus propiedades.

2.1.7.1-Definiciones y notación

Una red de Petri ordinaria se define como una cuádrupla $Q = \langle P, T, Pre, Post \rangle$ tal que:

$P = \{P_1, P_2, \dots, P_n\}$, es un conjunto de lugares, finito y no vacío;

$T = \{T_1, T_2, \dots, T_n\}$, es un conjunto de transiciones, finito y no vacío;

donde $P \cap T = \emptyset$, esto es que los conjuntos P y T son independientes.

$Pre: P \times T \rightarrow \{0,1\}$, es la matriz de incidencia de entrada.

$Post: P \times T \rightarrow \{0,1\}$, es la matriz de incidencia de salida.

$Pre(P_i, T_j)$ es el peso del arco que une P_i con T_j ; por lo tanto será 1, o el peso que tenga el arco, si el arco existe y 0 si no.

$Post(P_i, T_j)$ es el peso del arco que une T_j con P_i ; por lo tanto será 1, o el peso que tenga el arco, si el arco existe y 0 si no. Podemos notar que Pre y $Post$ están relacionados a la transición T_j y a cuáles son sus arcos de entrada o salida.

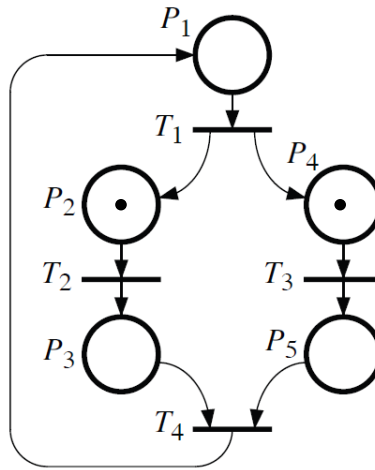


Figura 15.- Ejemplo de una red de Petri de la cual se calculan sus propiedades.

Otra forma de notación para referirnos a Pre y $Post$ es la siguiente:

$$\mathbf{W}^- = [w_{ij}^-], \text{ donde } w_{ij}^- = Pre(P_i, T_j) \quad \text{matriz de incidencia de entrada}$$

$$\mathbf{W}^+ = [w_{ij}^+], \text{ donde } w_{ij}^+ = Post(P_i, T_j) \quad \text{matriz de incidencia de salida}$$

$$\mathbf{W}^- = \begin{matrix} & \begin{matrix} T_1 & T_2 & T_3 & T_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{matrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{matrix} \end{matrix} \quad \mathbf{W}^+ = \begin{matrix} & \begin{matrix} T_1 & T_2 & T_3 & T_4 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \begin{matrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{matrix} \end{matrix} .$$

La siguiente matriz es conocida como matriz de incidencia:

$$\mathbf{W} = \mathbf{W}^+ - \mathbf{W}^- = [w_{ij}]$$

y para la red de Petri de la *figura* tenemos:

$$\mathbf{W} = \begin{matrix} & T_1 & T_2 & T_3 & T_4 \\ \begin{bmatrix} -1 & 0 & 0 & +1 \\ +1 & -1 & 0 & 0 \\ 0 & +1 & 0 & -1 \\ +1 & 0 & -1 & 0 \\ 0 & 0 & +1 & -1 \end{bmatrix} & P_1 \\ & P_2 \\ & P_3 \\ & P_4 \\ & P_5 \end{matrix}.$$

Cada columna de esta matriz corresponde a una modificación en el marcaje que se efectúa al disparar la transición correspondiente. Por ejemplo, la primera columna de **W** indica que el disparo de la transición T_1 consiste en remover un token de P_1 y agregar un token a los lugares P_2 y P_4 . Naturalmente esto no nos provee ninguna información concerniente a la posibilidad de disparar cierta transición ya que *la matriz de incidencia es independiente del marcaje*.

Es importante remarcar y notar que si una red de Petri es *pura*, es decir que no contiene bucles, entonces su matriz de incidencia puede ser encontrada. Pero si una rdP es *impura*, no se puede construir su matriz de incidencia ya que si se dispara la transición que contiene el bucle esto implica que se quiten y se agreguen tokens en el mismo lugar, por lo que implica una pérdida de información en la matriz de incidencia. Por lo tanto, toda la información contenida en una red de Petri puede ser conocida si se cuenta con su matriz de incidencia de entrada y de salida.

2.1.7.2.-Ecuación Fundamental

Sea S una secuencia de transiciones que pueden ser realizadas desde un marcaje \mathbf{m}_i . Por ejemplo, para el marcaje \mathbf{m}_1 de la figura 15, la secuencia de disparos $S_1 = T_2$ contiene a la transición T_2 una vez y a las otras transiciones ninguna. El **vector característico** de la secuencia S , escrito como \mathbf{s} , es el vector de m -componentes cuyo número de componente j corresponde al número de transiciones T_j en la secuencia S . Para este ejemplo sería $\mathbf{s}_1 = (0,1,0,0)$.

Si la secuencia de transiciones S es tal que se pueda ir de un \mathbf{m}_i hasta un \mathbf{m}_k mediante ella, entonces se puede obtener una **ecuación fundamental**:

$$\mathbf{m}_k = \mathbf{m}_i + \mathbf{W} \cdot \mathbf{s}$$

Para nuestro ejemplo de la figura 15:

$$\begin{matrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ \mathbf{m}_1 \end{matrix} + \begin{matrix} \begin{bmatrix} -1 & 0 & 0 & +1 \\ +1 & -1 & 0 & 0 \\ 0 & +1 & 0 & -1 \\ +1 & 0 & -1 & 0 \\ 0 & 0 & +1 & -1 \end{bmatrix} \\ \mathbf{W} \end{matrix} \cdot \begin{matrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \\ \mathbf{s}_1 \end{matrix} = \begin{matrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ \mathbf{m}_2 \end{matrix} + \begin{matrix} \begin{bmatrix} 0 \\ -1 \\ +1 \\ 0 \\ 0 \end{bmatrix} \\ \mathbf{m}_2 \end{matrix} = \begin{matrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \\ \mathbf{m}_2 \end{matrix} .$$

Otro ejemplo sería ver que marcaje obtenemos desde \mathbf{m}_2 si tenemos la siguiente secuencia de transiciones $S_2 = T_3T_4T_1T_3$, donde tendríamos $\mathbf{s}_2 = (1,0,2,1)$. Al realizar estas transiciones lo que obtenemos es el marcaje \mathbf{m}_3 ; como se puede ver a continuación:

$$\begin{matrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \\ \mathbf{m}_2 \end{matrix} + \begin{matrix} \begin{bmatrix} -1 & 0 & 0 & +1 \\ +1 & -1 & 0 & 0 \\ 0 & +1 & 0 & -1 \\ +1 & 0 & -1 & 0 \\ 0 & 0 & +1 & -1 \end{bmatrix} \\ \mathbf{W} \end{matrix} \cdot \begin{matrix} \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \end{bmatrix} \\ \mathbf{s}_2 \end{matrix} = \begin{matrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \\ \mathbf{m}_2 \end{matrix} + \begin{matrix} \begin{bmatrix} 0 \\ +1 \\ -1 \\ -1 \\ +1 \end{bmatrix} \\ \mathbf{m}_2 \end{matrix} = \begin{matrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\ \mathbf{m}_3 \end{matrix} .$$

Algo que resulta interesante notar, es que si consideramos una secuencia de transiciones $S = S_1S_2$ desde un marcaje \mathbf{m}_1 , tendríamos $S_1S_2 = T_2T_3T_4T_1T_3$, lo que es equivalente a tener $\mathbf{s}_1 + \mathbf{s}_2 = (1,1,2,1)$, lo que representado en la forma de la ecuación fundamental es $\mathbf{m}_1 + \mathbf{W} \cdot (\mathbf{s}_1 + \mathbf{s}_2) = \mathbf{m}_3$, lo cual se puede observar en la siguiente figura.

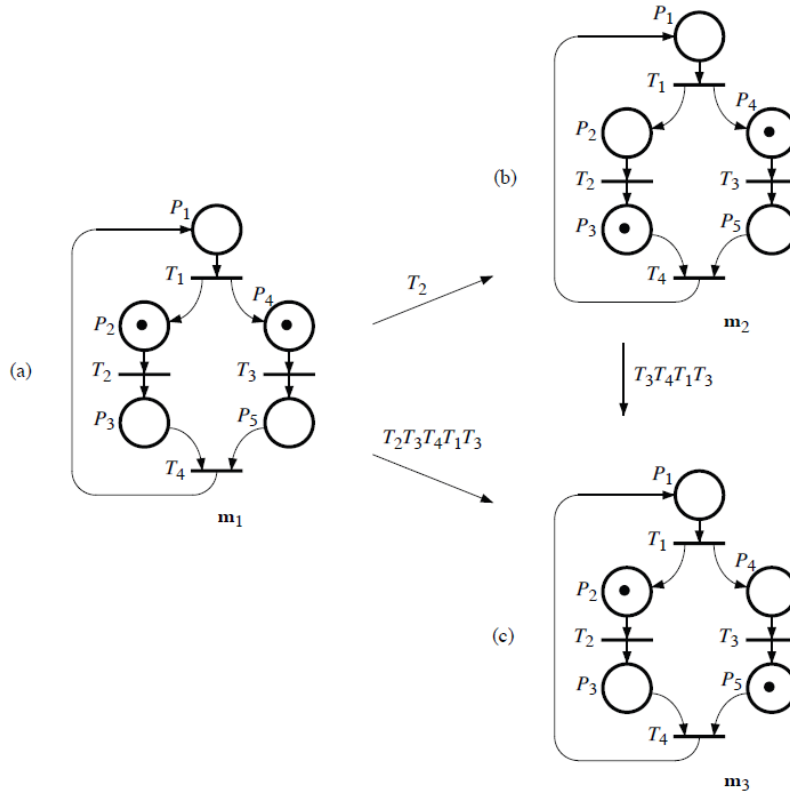


Figura 16.- Ejemplo de distintas secuencias de transiciones para una rdP.

Se dice que un vector s es un "posible" vector característico, si por lo menos una secuencia de disparos de transiciones S corresponde a él, desde un marcaje m_i . Por esta razón no todos los vectores s cuyas componentes son positivas o cero son posibles.

2.1.7.3.-Cálculo de invariantes

El vector x es un invariante de lugares si y solo si, satisface la siguiente ecuación:

$$x^T \cdot W = 0$$

La ecuación anterior resulta de la ecuación fundamental de la red, para un marcaje inicial m_0 desde el cual los marcajes alcanzables están descritos por:

$$m_k = m_0 + W \cdot s$$

La cual si es multiplicada por x^T por la izquierda de ambos términos de la ecuación obtenemos:

$$x^T \cdot m_k = x^T \cdot m_0 + x^T \cdot W \cdot s$$

Donde, si se cumple que $x^T \cdot W = 0$, el resultado es:

$$x^T \cdot m_k = x^T \cdot m_0$$

Lo que significa que para cualquier S el marcaje resultante es un marcaje invariante, ya que debido a que el resultado de $x^T \cdot m_0$ es un escalar. Lo que en otras palabras significa que el número de tokens en los lugares de la red, ponderados por x , es constante.

El vector x que resuelve la ecuación $x^T \cdot W = 0$, es conocido como P-Invariante (invariante de lugares). Solo se consideran como invariantes los valores no negativos, esto es $x \geq 0$.

El valor de estos invariantes nos permite conocer algunas propiedades de la rdP, como por ejemplo saber si la red es **conservativa**. Se dice que una rdP es conservativa si y solo si hay un P-invariante $x > 0$ tal que resuelva la ecuación $x^T \cdot W = 0$.

Por otro lado, si existe un P-invariante $x > 0$ tal que $x^T \cdot W \leq 0$ entonces la rdP es estructuralmente acotada.

Por ejemplo para la red de la figura 15, se tiene:

$$x_1^T \cdot W = [1 \ 1 \ 1 \ 0 \ 0] \cdot \begin{bmatrix} -1 & 0 & 0 & +1 \\ +1 & -1 & 0 & 0 \\ 0 & +1 & 0 & -1 \\ +1 & 0 & -1 & 0 \\ 0 & 0 & +1 & -1 \end{bmatrix} = [0 \ 0 \ 0 \ 0].$$

Lo que nos dice que los lugares $P(x_1) = \{P_1, P_2, P_3\}$ son componentes conservativas.

Existe otra propiedad que es similar a los P-invariantes. Un vector y que sea una solución a la expresión $W \cdot y = 0$, es conocido como un T-invariante (invariante de transiciones). El vector y considerado corresponde a un vector característico s asociado a una secuencia de transiciones S . Si $s = y$ entonces el conjunto de transiciones contenido en S , corresponde a los elementos que no son cero en s , ya que y debe estar formado por enteros positivos o ceros.

Por lo tanto si existe un vector característico s , tal que cumpla lo siguiente:

$$W \cdot s = 0$$

Entonces el vector s será un T-invariante.

Si $W \cdot s > 0$ entonces se tratará de una componente repetitiva.

Si $W \cdot s = 0$, entonces s es un T-invariante. Sin embargo no todos los T-invariantes necesariamente corresponden a una componente repetitiva, porque por lo menos una secuencia de disparos debe corresponder al vector s . Es interesante notar que el cálculo algebraico permite encontrar los T-invariantes (independientemente del marcaje). Por lo tanto de acuerdo a $W \cdot s = 0$, si hay un T-invariante s tal que S sea una secuencia de disparos desde un marcaje m , tal que se pueda ir de m hasta m por medio de s , entonces la existencia de tal T-invariante es necesariamente una condición para que la red de Petri sea **reinicializable**.

Se dice que una red de Petri es **consistente** si hay un T-invariante tal que $W \cdot y = 0$ si y solo si $y > 0$.

Si existe un $y > 0$ tal que $W \cdot y \geq 0$, entonces la rdP es **estructuralmente repetitiva**.

2.2.-Logica difusa (16) (11) (6)

Nuestra percepción del mundo está plagada de conceptos que no tienen límites bien definidos, por ejemplo, mucho, alto, mas grande que, joven, etc. Estos conceptos pueden ser llamados conceptos *difusos* o *grises* (vagos), el cerebro humano trabaja con ellos todo el tiempo, mientras que las computadoras quizás no lo hagan (ellas razonan con cadenas de 0's y 1's).

La lógica difusa fue presentada en 1965, por Lofti Zadeh, y es una herramienta matemática para tratar con la incertidumbre. Brinda una técnica para tratar con la imprecisión. La teoría difusa provee un mecanismo para representar conceptos lingüísticos como "mucho", "bajo", "medio", "frecuente", "algunos". En general, la lógica difusa provee una estructura de inferencia que permite apropiar capacidades del razonamiento humano. Por el contrario, la teoría binaria tradicional describe eventos concretos, eventos que u ocurren o no ocurren. Esta utiliza la probabilidad para explicar si un evento ocurrirá, midiendo la oportunidad con la que se espera que cierto evento ocurra. La teoría de la lógica difusa se basa en el conocimiento de un relativo grado de

pertenencia así como lo hacen las funciones del pensamiento y procesos cognitivos. La utilidad de los conjuntos difusos recae en su habilidad para modelar incertidumbre o ambigüedad en la información frecuentemente encontrada en la vida diaria.

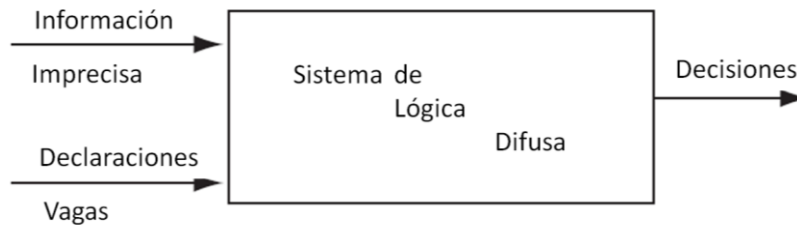


Figura 17.- Representación de un sistema difuso simple.

Al hablar sobre difuso se refiere a que describe la ambigüedad de un evento y la aleatoriedad describe la incertidumbre de que ocurra un evento. Generalmente, puede observarse en la teoría clásica de conjuntos que no existe la incertidumbre, debido a que tienen límites bien definidos, pero en el caso de conjuntos difusos, ya que la incertidumbre existe, los límites pueden estar ambiguamente especificados.

Los conjuntos difusos son los bloques usados para formar un sistema de reglas IF_THEN, las cuales tienen generalmente la forma "Si x es A entonces y es B " donde A y B son conjuntos difusos. El término sistemas difusos se refiere principalmente a sistemas que están gobernados por reglas difusas IF_THEN. La parte del IF es una implicación y es llamada el antecedente, mientras que la segunda parte, THEN es una consecuencia. Un sistema difuso es un conjunto de reglas difusas que convierten entradas en salidas.

La desventaja de los sistemas difusos son sus reglas; reglas inteligentes generan sistemas inteligentes y otras reglas generan sistemas menos inteligentes o incluso sistemas tontos. El número de reglas crece exponencialmente con la dimensión del espacio de entrada, es decir el número de entradas del sistema. A esto se le llama el principio de dimensionalidad y es un problema general para los modelos matemáticos.

Por lo tanto, los modelos difusos no son un reemplazo para los modelos probabilísticos. En algunos casos, los modelos difusos resultan funcionar mejor pero en otros caso no. Pero principalmente han probado evidentemente brindar una mejor solución para problemas complejos.

2.2.1.-Conjuntos difusos

En la teoría de conjuntos clásica, su función característica asigna un valor ya sea de 1 o 0 a cada individuo en el conjunto universal, para de esta forma discriminar entre los miembros y los no miembros del conjunto clásico que se está considerando. Los valores asignados a los elementos del conjunto universal caen dentro de un rango establecido e indican el grado de pertenencia de estos elementos a un conjunto. Valores más grandes denotan un mayor grado de pertenencia al conjunto, la función que describe este comportamiento de pertenencia a un conjunto, es llamada función de membresía y el conjunto al cual describe, es llamado conjunto difuso.

Un conjunto difuso es por lo tanto una colección de elementos con diversos grados de membresía referentes al conjunto. Esta idea está en contraste con los conjuntos clásicos, porque los miembros de estos conjuntos no serán miembros a menos que su membresía esté completa, es decir que tenga un valor de 1. Los elementos de un conjunto difuso, dado que su membresía no necesita estar completa, pueden ser miembros de otros conjuntos difusos en el mismo universo. Para denotar a los conjuntos difusos se usará una letra subrayada, ejemplo \underline{A} . Los conjuntos difusos se mapean a un valor en los reales con un intervalo entre 0 y 1. Si un elemento del universo, por decir x , es miembro del conjunto \underline{A} , entonces el mapeo de sus valores estará dado por $\mu_{\underline{A}}(x) \in [0,1]$. A esto se le llama grado de pertenencia.

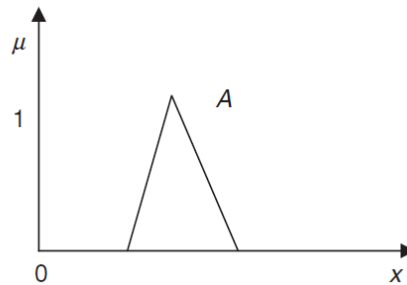


Figura 18.- Función de membresía que indica el grado de pertenencia.

Formalmente un conjunto difuso \underline{A} en X es expresado como un conjunto de pares ordenados:

$$\underline{A} = \{(x, \mu_{\underline{A}}(x)) \text{ donde } x \in X\}$$

El par ordenado también se expresa como $x/\mu_{\underline{A}}(x)$.

2.2.2.- Características de las funciones de membresía

Las características de las funciones de membresía se definen mediante tres propiedades las cuales son:

- Núcleo
- Soporte
- Límite

La función de membresía puede tomar valores entre 0 y 1, las propiedades anteriores se muestran en la siguiente figura.

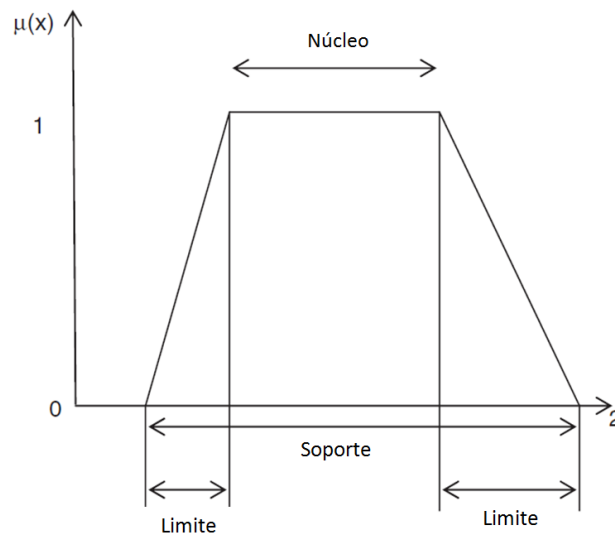


Figura 19.- Partes de una función de membresía.

1. **Núcleo:** Son los elementos de la función de membresía que tienen un valor igual a 1, esto es $\mu_{\underline{A}}(x) = 1$.
2. **Soporte:** Son los elementos cuyo grado de membresía es mayor que 0, esto es $\mu_{\underline{A}}(x) > 0$.
3. **Límite:** Son los elementos que tienen un grado de membresía entre 0 y 1, es decir $0 < \mu_{\underline{A}}(x) < 1$.

Es importante definir dos términos más relacionados con las funciones de membresía:

- *Punto de cruce*: Son los elementos que tienen un grado de membresía igual a 0.5, $\mu_{\underline{A}}(x) = 0.5$.
- *Altura*: La altura del conjunto difuso \underline{A} será el valor máximo de la función de membresía, $\max(\mu_{\underline{A}}(x))$.

2.2.3.- Fusificación

El concepto de fusificación es un concepto importante en la teoría de la lógica difusa. La fusificación es el proceso donde los límites bien definidos se convierten en límites difusos. Al identificar algunas de las incertidumbres presentes es como se eligen los valores difusos. La conversión a valores difusos está representada por las funciones de membresía.

En cualquier aplicación práctica, en la industria, investigaciones, mediciones de voltaje, corriente, temperatura, etc., tal vez existirá un error inevitable quizás insignificante. El cual causa imprecisión en la información. Esta imprecisión puede ser representada por las funciones de membresía. Por lo tanto, se realiza una fusificación.

2.2.4.- Asignación de funciones de membresía

Existen varios métodos para asignar las funciones de membresía para las variables difusas. Esta designación puede realizarse solo por intuición o mediante algún algoritmo o procedimientos lógicos. Algunos de los métodos para asignar las funciones de membresía son los siguiente:

- Intuición
- Inferencia
- Ordenamiento mediante clasificación (Rank ordering)
- Conjuntos difusos angulares
- Redes neuronales
- Algoritmos genéticos
- Razonamiento inductivo

2.2.4.1.- Intuición

Se basa en la propia inteligencia y entendimiento de la persona para desarrollar las funciones de membresía. Se requiere de un conocimiento profundo del problema, además de contar con el conocimiento referente a sus variables lingüísticas.

Por ejemplo, considere la velocidad de un motor de corriente directa (DC). El universo de discurso es definido en rpm como se muestra en la figura:

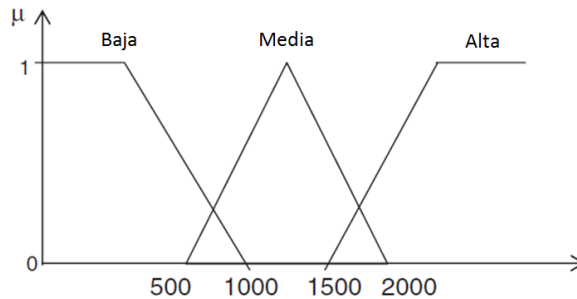


Figura 20.- Conjuntos difusos que representan las velocidad de giro en rpm de un motor.

Las curvas que representan las funciones de membresía corresponden a las variables difusas. El rango de velocidad es dividido en bajo, medio y alto. Las curvas diferencian los rangos elegidos por la persona. El lugar de las curvas es aproximado sobre el universo de discurso, el número de curvas y la superposición entre ellas es un criterio importante que debe considerarse mientras se definen las funciones de membresía.

2.2.4.2.- Inferencia

Este método involucra el conocimiento para realizar un razonamiento deductivo. Las funciones de membresía se forman de los hechos y el conocimiento del problema.

Usando el método de inferencia se puede identificar un triángulo. Sea U el universo de triángulos y A , B y C los ángulos internos de los triángulos. También sabemos que $A \geq B \geq C \geq 0$. Por lo tanto, el universo U está descrito por:

$$U = \{(A, B, C), A \geq B \geq C \geq 0, A + B + C = 180^\circ\}$$

Hay diferentes tipos de triángulos:

\underline{I} = Isósceles \underline{R} = Rectángulo \underline{O} = Otros

El grado de membresía puede ser inferido para todas estas clases de triángulos, ya que se tiene el conocimiento sobre su geometría.

La función de membresía para los triángulos isósceles está dada por:

$$\mu_{\underline{I}}(A, B, C) = 1 - \frac{1}{60^\circ} \min(0, A - B, B - C)$$

Esto es debido a que se sabe que para que un triángulo sea isósceles debe cumplirse que dos de sus ángulos internos sean iguales, es decir que $A = B$ o $B = C$.

La función de membresía para los triángulos rectángulos está dada por:

$$\mu_{\underline{R}}(A, B, C) = 1 - \frac{1}{90^\circ} \min(0, A - 90^\circ, B - 90^\circ, C - 90^\circ)$$

Se define de esta forma, porque se sabe que un triángulo rectángulo es aquel que uno de sus ángulos internos es igual a 90° .

Como es posible realizar operación lógicas con las funciones de membresía, entonces se puede expresar una función de membresía usando la unión o la intersección de otras previamente definidas, de esta forma obtenemos lo siguiente:

$$\mu_{\underline{O}}(A, B, C) = \overline{\mu_{\underline{I}}} \cap \overline{\mu_{\underline{R}}} = \min(1 - \mu_{\underline{I}}(A, B, C), 1 - \mu_{\underline{R}}(A, B, C))$$

Porque el conjunto de otros triángulos está formado por todos los triángulos que no son ni isósceles, ni tampoco rectángulos.

2.2.4.3.- Funciones de membresía comúnmente usadas

Una función de membresía triangular se define como:

$$f(x; a, b, c) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right)$$

Una función de membresía de gauss se define como:

$$f(x; c, \sigma) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}$$

Una función de membresía campana de gauss se define como:

$$f(x, a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}$$

Una función de membresía sigmoide se define como:

$$f(x, a, c) = \frac{1}{1 + e^{-a(x-c)}}$$

2.2.5.- Defusificación

Quiere decir convertir un valor difuso a un valor bien limitado. Los resultados difusos no pueden ser utilizados tal cual para las aplicaciones, por lo tanto es necesario convertirlos de grados de membresía a valores cuantitativos relacionados al universo de discurso del problema para un subsecuente procesamiento. Esto se logra mediante un proceso de defusificación. La defusificación tiene la capacidad de reducir los valores difusos a una cantidad con un solo valor, es decir que reduce la colección de valores de las funciones de membresía a una cantidad única.

2.2.5.1.- Corte lambda

Considerando un conjunto difuso \underline{A} , entonces el corte lambda de este conjunto se puede denotar por A_λ , donde el rango de lambda está entre 0 y 1 ($0 \leq \lambda \leq 1$).

El conjunto A_λ será un conjunto con limites bien definidos por:

$$A_\lambda = \left\{ x / \mu_{\underline{A}}(x) \geq \lambda \right\}$$

Donde x será el valor del conjunto de corte lambda, cuando el valor de la función de membresía en x , $\mu_A(x)$, sea mayor o igual que el λ especificado. Al corte lambda suele llamársele también corte alfa.

2.2.5.2.- Métodos de defusificación

Además de los cortes lambda que convierten a los conjuntos difusos en conjuntos clásicos, existe una variedad de métodos de defusificación que pueden ser empleados para convertir los valores difusos a cantidades únicas. Estos métodos son usados debido a que usualmente la salida de un sistema difuso está compuesta por la unión o la intersección de dos o más funciones de membresía como se muestra en la figura 21.

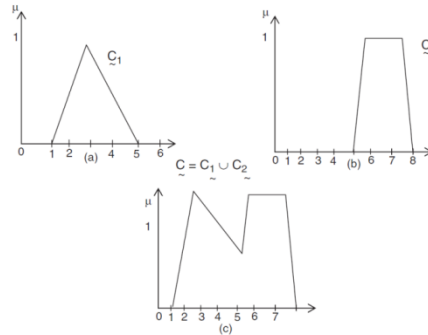


Figura 21.- Unión de dos conjuntos difusos que representa la salida de un sistema.

Algunos de los métodos que existen para defusificar las funciones difusas de salida son:

- Método de la altura: En este método se toma el valor z^* para cuando $\max(\mu_C(z))$, es decir el valor de z que haga máximo el valor del grado de pertenencia.

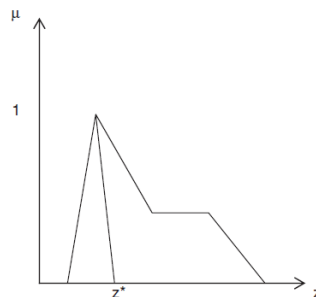


Figura 22.- Defusificación por método de altura.

- Método del centroide: Es el método más usado. También es llamado método del centro de gravedad o método del centro de área. Se define de manera algebraica como:

$$z^* = \frac{\int z \mu_{\underline{C}}(z) dz}{\int \mu_{\underline{C}}(z) dz}$$

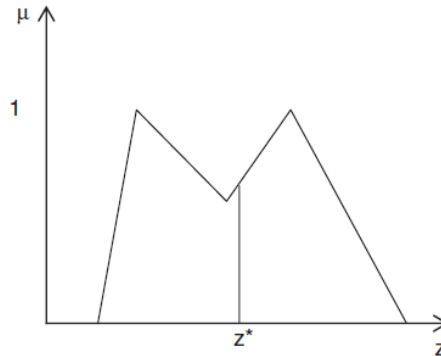


Figura 23.- Defusificación por método del centroide.

- Método de promedio ponderado: Este método solo puede ser usado para funciones de membresía de salida que sean simétricas. Este método consiste en ponderar cada función de membresía de salida entre su valor más grande de grado de pertenencia.

$$z^* = \frac{\sum \mu_{\underline{C}}(z) z}{\sum \mu_{\underline{C}}(z)}$$

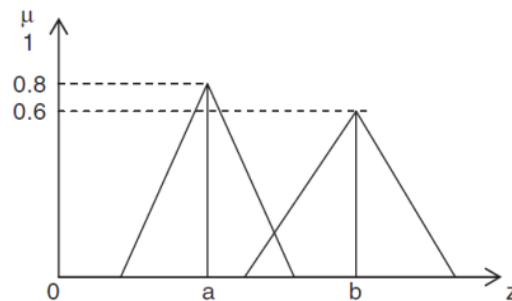


Figura 24.- Defusificación por método del promedio ponderado.

- Método del máximo medio: Esta definido por el rango de valores máximos que posea la función de membresía, el valor máximo no debe ser un punto único, se puede expresar de la siguiente forma:

$$z^* = \frac{a + b}{2}$$

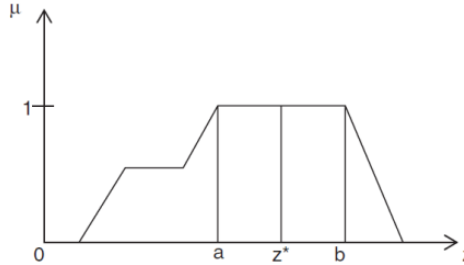


Figura 25.- Defusificación por máximo medio.

2.2.6.- Sistemas difusos de inferencia

Los sistemas difusos de inferencia (FIS) también son conocidos como sistemas difusos basados en reglas, modelos difusos, sistemas difusos expertos y memorias asociativas difusas. La toma de decisiones es una parte importante en todo el sistema. Los FIS formulan reglas apropiadas y basándose en las reglas toman la decisión. Están basados principalmente en los conceptos de los de los conjuntos difusos, las reglas IF_THEN y el razonamiento difuso. Los FIS usan declaraciones "Si .. entonces" junto con los conectores "O" o "Y" para formar las reglas necesarias para la toma de decisiones.

2.2.6.1.- Construcción y funcionamiento de un sistema difuso de inferencia

Los sistemas difusos de inferencia consisten en una interfaz de fusificación, una base de reglas, una base de datos, una unidad de toma de decisiones, y finalmente una interface de defusificación. Un FIS formado por cinco bloques puede verse en la *figura*. La función de cada bloque es la siguiente:

- **Base de reglas:** Contiene una colección de reglas difusas IF_THEN.
- **Base de datos:** Define las funciones de membresía de los conjuntos difusos usados en las reglas.
- **Unidad de toma de decisiones:** Realiza las operaciones de inferencia sobre las reglas.
- **Interfaz de fusificación:** Transforma los valores de las entradas a grados de pertenencia relacionados a las variables lingüísticas.
- **Interfaz de defusificación:** Transforma los valores difusos a valores únicos.

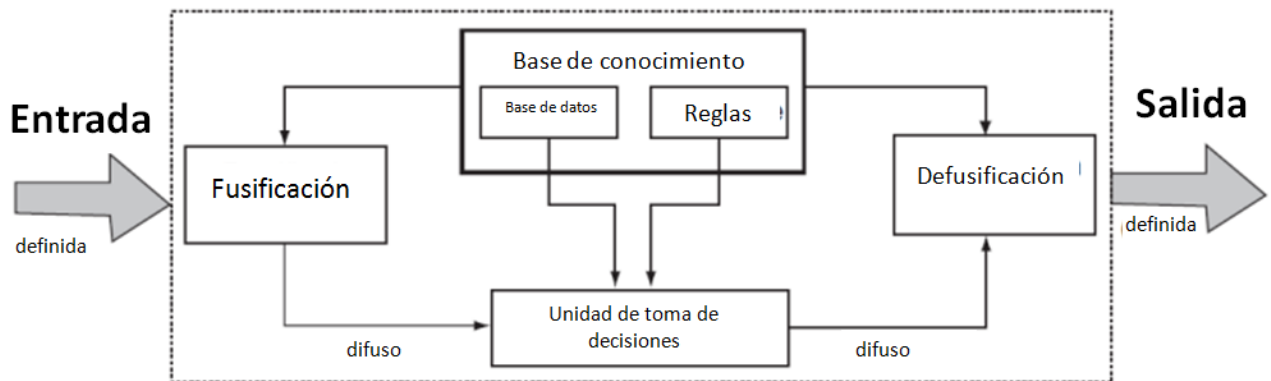


Figura 26.- Modelo de un sistema difuso de inferencia (FIS).

El funcionamiento del FIS es el siguiente. La entrada se convierte a difuso usando un método de fusificación. Después de la fusificación se forma la base de reglas. A la base de datos y la base de reglas juntas son conocidas como la base de conocimiento del sistema. La defusificación es usada para convertir los valores difusos a valores del mundo real para que sean la salida del sistema.

Los pasos del razonamiento difuso (las operaciones de inferencia sobre las reglas IF_THEN) realizados por el FIS son:

1. Evaluar los valores de la entrada en las funciones de membresía que pertenecen a los antecedentes para obtener valores difusos para cada etiqueta lingüística.
2. Combinar (mediante un operador t-norma específico) los valores de membresía en los antecedentes para obtener la fuerza de disparo (peso) de cada regla.
3. Encontrar los consecuentes generados para cada regla dependiendo de los pesos calculados.
4. Aplicar un método de defusificación a la suma de los consecuentes calculados de las reglas para obtener la salida del sistema.

2.2.7.- Métodos difusos de inferencia

Los dos métodos más importantes son el método difuso de inferencia de Mamdani, el cual es comúnmente utilizado. Este método fue presentado por Mamdani y Assilian en 1975. Otro método de inferencia muy conocido es el llamado Sugeno o Takashi-Sugeno-Kang. Este método fue presentado por Sugeno en 1985 también se suele referirse a él como método TS. La principal diferencia entre estos dos métodos recae en los consecuentes de las reglas difusas. Los sistemas difusos Mamdani usan conjuntos difusos como consecuencias de las reglas mientras que los sistemas difusos TS utilizan funciones lineales de las variables de entrada como consecuentes de las reglas. A continuación se detalla el funcionamiento de los sistemas difusos Mamdani el cual fue empleado en este trabajo.

2.2.8.- Método difuso de inferencia Mamdani

El método difuso de inferencia Mamdani es el método más común, estuvo entre los primeros sistemas que fueron construidos usando la teoría de conjuntos difusos. Fue propuesto por Mamdani en 1975 como un intento para controlar una combinación de motor de vapor con un boiler mediante la síntesis de un conjunto de reglas lingüísticas de control obtenidas de la experiencia de los operadores. Los esfuerzos de Mamdani se basaron en el artículo de 1973 de Zadeh sobre algoritmos difusos para sistemas complejos y procesos de decisión.

Un ejemplo de un sistema de inferencia tipo Mamdani se muestra en la *figura*. Para calcular la salida de un FIS dadas dos entradas, se deben seguir seis pasos:

1. Determinar un conjunto un conjunto de reglas difusas.
2. Convertir las entradas mediante las funciones de membresía.
3. Combinar las entradas difusas de acuerdo a las reglas para determinar los pesos de las reglas.
4. Encontrar el consecuente de las reglas al combinar los pesos de las reglas y las funciones de membresía.
5. Combinar los consecuentes para obtener una distribución de salida.
6. Defusificar la distribución de salida.

Lo siguiente es una descripción más detallada del proceso.

2.2.8.1.- Creación de las reglas difusas

Las reglas difusas son una colección de declaraciones lingüísticas que describen como el FIS, debe tomar una decisión con respecto a sus entradas o para controlar la salida. Las reglas difusas siempre se escriben de la siguiente forma:

Si (la entrada 1 es la función de membresía 1) **y/o** (la entrada 2 es la función de membresía 2) **y/o** ... **entonces** (salida n es la función de membresía de salida n).

Por ejemplo:

Si la temperatura es alta **y** la humedad es alta **entonces** la habitación esta calurosa.

Debe haber una función de membresía que determine temperatura alta (entrada 1), humedad alta (entrada 2) y habitación calurosa (salida 1). El proceso de tomar una entrada y convertirlo en valores de una función de membresía es llamado fusificación, también es necesario definir las operaciones o combinaciones relacionadas a los conectores **y/o**. Se utiliza el operador mínimo para calcular el "**y**" difuso que combina dos funciones de membresía para calcular el peso de la regla. Para el operador "**o**" difusos se utiliza el operador máximo de igual forma para combinar las funciones de membresía.

2.2.8.2.- Fusificación

El propósito de la fusificación es mapear las entradas de un sensor (o de sus características como puede ser la amplitud o el espectro) a valor entre 0 y 1 mediante un conjunto de funciones de membresía. Estas funciones de membresía de entrada, pueden representar conceptos difusos como "grande" o "pequeño", "viejo" o "joven", "caliente" o "frio", etc. Cuando se eligen las funciones de membresía la definición de grande o pequeño puede ser diferente entre cada entrada.

2.2.8.3.- Consecuentes

Los consecuentes de una regla difusa con calculados usando dos pasos:

1. Calcular el peso de las reglas al combinar las funciones de membresía de entrada mediante los operadores "**y**"(min) "**o**" (max).

2. Realizarle un corte lambda a la función de membresía de salida de acuerdo al peso de la regla.

2.2.8.4.- Distribución de salida y defusificación

Las salidas de todas las reglas difusas deben combinarse para obtener una distribución difusa de salida. Esto usualmente se obtiene al usar el operador difuso "o".

En muchas ocasiones se desea tener un solo valor de salida del FIS. Esta cantidad se obtiene al aplicar un método de defusificación a la distribución difusa de salida. Algunas técnicas de defusificación ya fueron descritas previamente y pueden ser usadas para obtener la salida del sistema difuso de inferencia Mamdani.

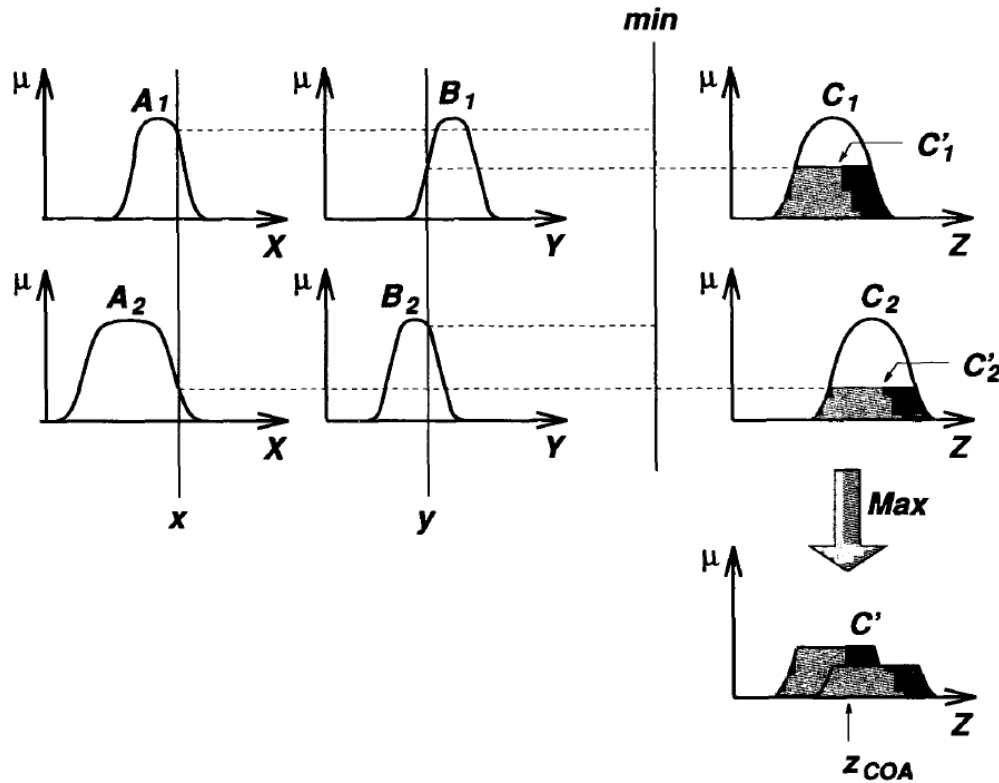


Figura 27.- Ejemplo de un sistema difuso de inferencia Mamdani.

Capítulo 3

Plataforma de Prueba

En este capítulo se explican las características del robot que fue usado para probar el desempeño del sistema híbrido propuesto, así como la forma en que se puede programar. También se menciona el protocolo de comunicación que se utiliza para comunicar al robot con la computadora y de igual forma se mencionan las características y se describe el funcionamiento de los actuadores y sensores utilizados en este trabajo.

3.1.-Bioloid Kit Premium (17)

Es un kit de robótica didáctica fabricado por la empresa coreana ROBOTIS, la cual fabrica una gran gama de productos, desde servo-motores pasando por robots de mayor complejidad como el DARwin-OP e incluso kits de robótica más sencillos que el Bioloid.

El Biolod kit premium está compuesto por 18 servomotores de la marca DYNAMIXEL, también fabricados por la compañía ROBOTIS, cuenta con una unidad central de procesamiento denominada CM-510, la cual en su interior contiene un micro controlador AVR de 32 bits que es el encargado de la comunicación del robot mediante el protocolo serial, controlar el giro de los servomotores y administrar los puertos que son convertidores analógico a digital para conectar en ellos los sensores con los que cuenta que son: un giroscopio, y un sensor infrarrojo para medir distancia, además de esto tiene un gran número de piezas plásticas con las cuales se puede ensamblar en distintas configuraciones, como un dinosaurio, una araña, un perro, entre muchas otras donde se incluye la posibilidad de ensamblarlo como un humanoide, que es la manera en que se uso para este trabajo.



Figura 28.- Contenido del Bioloid kit Premium.

En la configuración de robot humanoide existen 3 maneras distintas de armarlo, la diferencia consiste en el número de grados de libertad con los que cuenta el robot. Se eligió la configuración A sobre las otras dos alternativas debido a que brinda una mayor estabilidad al caminar y para realizar giros, esto debido a que cuenta con un mayor número de grados de libertad en la cadera. De ahora en adelante nos referiremos al BIOLOID kit premium en la configuración de humanoide tipo A, como el *robot bioloid*.

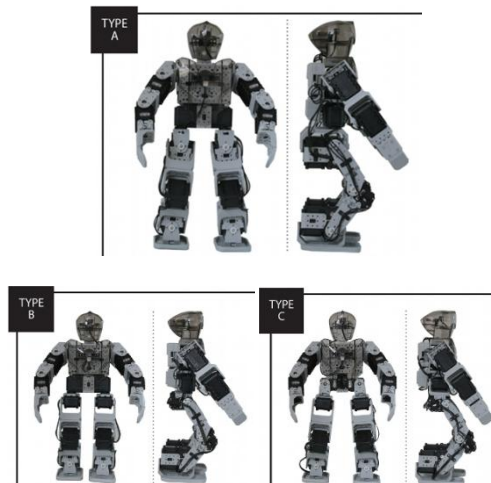


Figura 29.- Configuraciones tipo humanoide del robot bioloid.

La unidad CM-510 que es la encargada del procesamiento de información y del control de todos los periféricos del *robot bioloid*, tiene la siguiente configuración de puertos de entrada y salida:

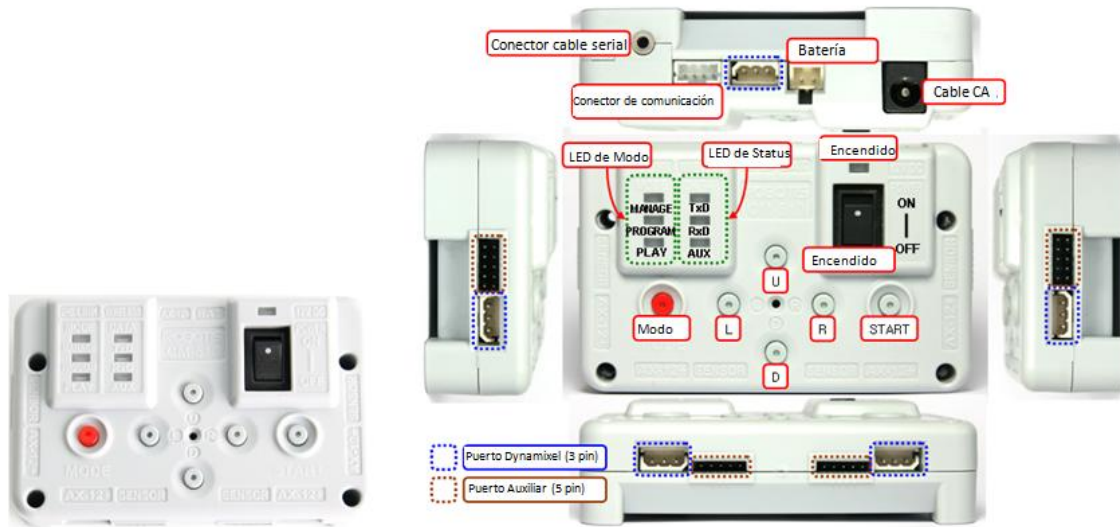
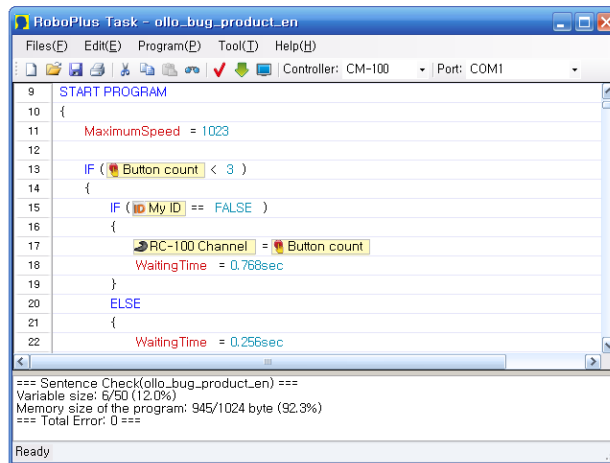


Figura 30.- Unidad central de procesamiento del robot bioloid (CM-510).

- **Conector cable serial:** Usado para conectar mediante comunicación serial al CM-510 y la PC para descargar códigos o transferencia de información.
- **Conector de comunicación:** Usado para la comunicación inalámbrica mediante el protocolo ZigBee u otros módulos externos. Básicamente este puerto es otra conexión de un puerto serial del microcontrolador para comunicarlo con algún otro periférico.
- **Botón de modo:** Se usa para cambiar la configuración del modo de operación del CM-510. Entre las cuales existen tres manager, programar y activado. Cada uno de estos se usa dependiendo de qué tarea se esté haciendo con el *robot bioloid*. Para ejecutar los programas que estén cargados en el CM-510 es necesario que el modo este en Activado.
- **Puerto Dynamixel:** En estos puertos se conectan los motores, que entre ellos tienen una conexión de tipo serie.
- **Puerto auxiliar:** En este puertos se conectan los sensores de distancia, giroscopios o cualquier otro sensor analógico que desee utilizarse.

Existen varias alternativas para trabajar y operar al *robot bioloid* las principales son mediante un programa que proporciona el fabricante, llamado RoboPlus, que contiene las siguientes aplicaciones:

RoboPlus Task: Es un programa para crear y descargar programas al robot. Aquí se pueden desarrollar programas en un lenguaje similar a C, el cual cuenta con una serie de elementos y funciones propias del lenguaje, como algunas especiales para controlar la posición de los motores o para que el robot ejecute movimientos previamente cargados mediante la aplicación RoboPlus Motion, también se pueden utilizar variables y ciclos así como crear funciones e invocarlas durante el código. La idea al escribir un programa para el robot en esta aplicación es crear una serie de tareas o movimientos de los motores de manera individual y secuencial, también se pueden usar los valores obtenidos del convertidor analógico digital, es decir las lecturas obtenidas mediante los sensores para condicionar las tareas que se realizan. Para descargar los programas generados es necesario que el modo del CM-510 este en programar.



```
RoboPlus Task - ollo_bug_product_en
Files(E)  Edit(E)  Program(P)  Tool(T)  Help(H)
Controller: CM-100  Port: COM1

9  START PROGRAM
10 {
11     MaximumSpeed = 1023
12
13     IF ( Button count < 3 )
14     {
15         IF ( My ID == FALSE )
16         {
17             RC-100 Channel = Button count
18             WaitingTime = 0.768sec
19         }
20     ELSE
21     {
22         WaitingTime = 0.256sec
23     }
24 }

=== Sentence Check(ollo_bug_product_en) ===
Variable size: 6/50 (12.0%)
Memory size of the program: 945/1024 byte (92.3%)
=== Total Error: 0 ===

Ready
```

Figura 31.- Ejemplo de la pantalla del programa RoboPlus Task.

RoboPlus Motion: Es un programa para crear y descargar movimientos al robot. Esta aplicación nos permite generar movimientos para el robot, como es caminar, mover los brazos u otros movimientos complejos de manera más simple que hacerlo mediante el control del giro de cada uno de los motores. Para hacer eso el programa cuenta con una interfaz gráfica en la cual se puede tener una representación virtual de la configuración física del robot, en este caso un humanoide, y mover sus articulaciones de la manera en que se desee, para crear una secuencia de posiciones que al ejecutarse una tras otra formen un movimiento complejo, como que el robot de pasos, se puede entender como generar el movimiento en un video cuadro por cuadro usando de base una secuencia de fotografías. Este programa es una ayuda o un recurso para desarrollar actividades más complejas con el *robot bioloid* ya que para acceder a los movimientos cargados en el robot es necesario hacerlo mediante los métodos que están disponibles en el programa RoboPlus task. Para descargar los programas generados es necesario que el modo del CM-510 este en programar.

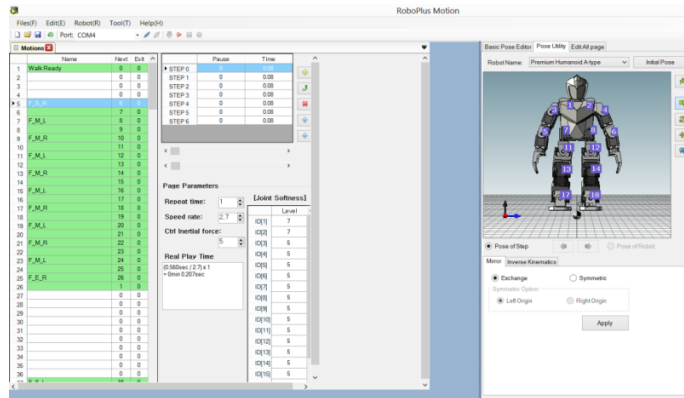


Figura 32.- Ejemplo del programa RoboPlus Motion.

RoboPlus Manager: Este software nos permite configurar los componentes del robot, como es los identificadores de cada motor, la velocidad y la posición inicial de los motores, la velocidad de transmisión de datos mediante el protocolo serial y la configuración de la comunicación ZigBee entre otras cosas. También otra de sus funciones es descargar e instalar las actualizaciones de software y el firmware del *robot bioloid* que proporciona el fabricante. Su función principal es la de probar los componentes de manera individual para buscar fallas o conocer el estado en que se encuentran. Para utilizar este programa es necesario que el modo del CM-510 este en manager.

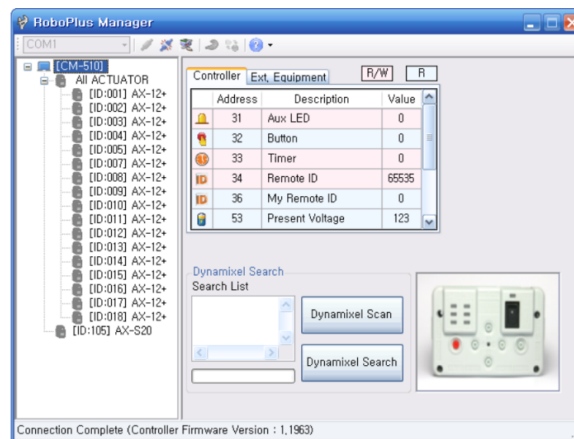


Figura 33.- Ejemplo del programa RoboPlus Manager.

C Embebido: Otra alternativa para utilizar al *robot bioloid* es usando C para microcontroladores y crear el código en AVR Studio, para esto el proveedor brinda la documentación de conexiones y administración de los puertos así como algunas librerías para el uso de la comunicación ZigBee. Hay que mencionar que esta manera nos da total control del *robot bioloid* ya que se tiene acceso a todos sus periféricos y a todos sus procesos internos, interrupciones y la administración de los recursos de memoria del microcontrolador.

3.2.-Módulo ZigBee (18)

Como ya se menciona el robot cuenta con una comunicación serial para que de esta manera se descarguen los programas, para ello es necesario un convertidor de serial a USB para conectar al robot con la computadora, pero existe la posibilidad de hacer esta comunicación de manera inalámbrica mediante el protocolo de comunicación ZigBee el cual se basa en radio frecuencia, si se utiliza esta forma de conexión no es posible descargar programas en el robot *bioloid* mediante este protocolo, pero nos da la posibilidad de establecer una comunicación entre el microcontrolador del robot y la computadora para un envío de datos bidireccional y de esta manera controla el motor o monitorear las condiciones del robot desde la computadora.

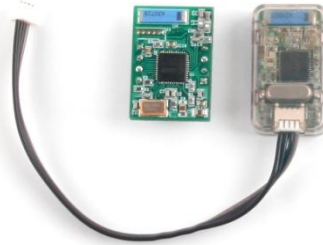


Figura 34.- Módulo emisor y receptor Zigbee ZIG-100/110-A.

Para la comunicación Zigbee el *robot bioloid* utiliza los módulos ZIG-100/110-A que son unos pequeños módulos que tiene un MCU y un Zigbee IC, que permiten una conexión inalámbrica usando la frecuencia de 2.4Ghz. Se utiliza como un módulo PAN (Red de Área Personal), el cual reemplaza a la comunicación serial cableada. Las especificaciones de estos módulos son las siguientes:

- Un ancho de banda de transferencia de datos por radiofrecuencia de 250 kbps máximo.
- Interface por medio de UART, que puede ser configurada para enviar y recibir información. Soporta un rango de baudios entre 9600 bps y 115200 bps.
- Un voltaje de alimentación de 3.3 V con un consumo de corriente de 30 mA.
- Dimensiones de 26.5 mm de ancho, 19 mm de largo y 12 mm de altura.

El siguiente diagrama representa la función que realiza este módulo de comunicación inalámbrica:

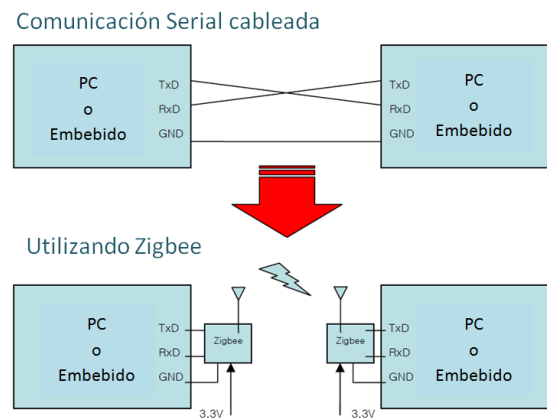


Figura 35.- Diagrama del funcionamiento de la comunicación Zigbee.

Para conectar el módulo Zigbee a la computadora es necesario contar con un adaptador el cual se muestra en la figura 36. El proveedor brinda las bibliotecas y archivos necesarios para poder desarrollar programas en Visual Basic, C++ y C# que puedan utilizar la comunicación Zigbee para enviar y recibir información del *robot bioloid* hacia la computadora y viceversa.

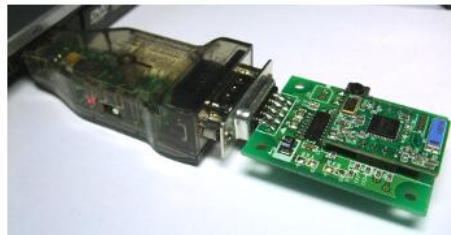


Figura 36.- Adaptador para comunicación Zigbee con la computadora.

3.3.-Motor Dynamixel AX-12

El motor AX-12 Dynamixel es actuador modular inteligente que incorpora un reductor de engranes, un motor de precisión de corriente directa y un circuito de control que tiene funcionalidades de trabajar en red con otros motores. A pesar de su reducido tamaño,

puede producir un torque elevado y es capaz de resistir grandes fuerzas externas debido a los materiales con los que es fabricado. Además cuenta con la habilidad de detectar y actuar de acuerdo a sus condiciones internas como pueden ser cambios en la temperatura interna o el voltaje de alimentación. Puede ser controlado mediante su posición o su velocidad con una resolución de 1024 pasos, que es equivalente a $.35^\circ$ por paso. Cuenta con una retroalimentación de su posición angular, velocidad angular y de la carga a la que esté sometido el motor.



Figura 37.- Motor Dynamixel AX-12.

Tabla 1.- Especificaciones del actuador AX-12.

Peso	55 gramos	
Razón de reducción de los engranes	1/254	
Voltaje de entrada	7 V	10 V
Torque máximo	12 kgf cm	16.5 kgf cm
Resolución	35°	
Angulo de operación	300° , giro completo	
Corriente máxima	900 mA	
Temperatura de operación	-5 °C a 85 °C	
Señal de comunicación	Paquete digital de datos	
Protocolo	Serial	

3.4.-HAVIMO 2.0

Es un módulo de procesamiento de imágenes que está equipado con una cámara CMOS y un microcontrolador que realiza el procesamiento de la imagen y los resultados son obtenidos mediante un puerto serial. La HAVIMO 2.0 está equipada con un algoritmo de procesamiento de imágenes el cual están enfocados a la detección de color, la cámara puede detectar hasta 16 áreas de distintos colores.

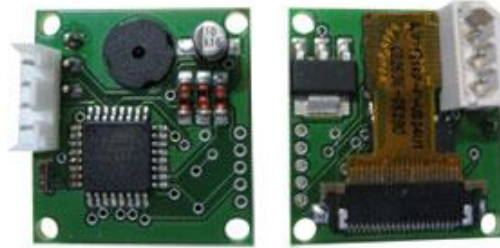


Figura 38.- Cámara HAVIMO 2.0.

Las especificaciones de la cámara HAVIMO 2.0 son las siguientes:

- Resolución de 160 * 120 píxeles.
- Resolución de color de 12 bits YCrCb.
- 19 Cuadros por segundo.
- Compatible con ROBOTIS CM-510.
- Compatible con software RoboPlus.
- Dimensiones 25 x 30 mm

Para la detección de color antes es necesario realizar una calibración, para la cual existe una interfaz GUI que facilita la elección de los colores que reconocerá, es necesario conectar la cámara con la computadora, pero debido a que la cámara HAVIMO es totalmente compatible con la unidad de procesamiento CM-510 del *robot bioid* solo es necesario conectar la cámara en uno de los puertos Dynamixel para poder utilizarla y conectar el CM-510 a la computadora para poder realizar la calibración.

La figura 39 muestra la pantalla de la interfaz GUI, la cual una vez que ya se estableció la comunicación entre la cámara y la computadora nos da la posibilidad de tomar una foto y de esta imagen seleccionar el color que queremos calibrar, el cual será el que busque la cámara mediante su algoritmo de procesamiento de imágenes.

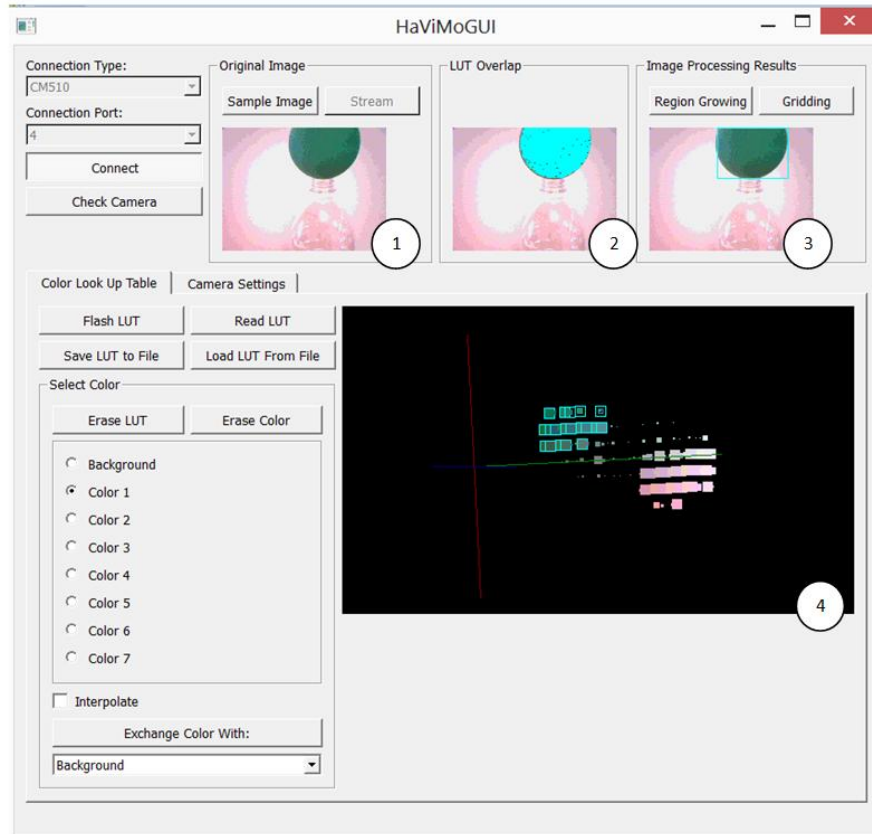


Figura 39.- Interfaz de calibración HAVIMO-GUI

1. Esta es la imagen obtenida por la cámara, para transferirla a la computadora tarda algunos segundos, por lo que esta cámara no es adecuada para enviar fotos en tiempo real a la computadora para un post-procesamiento.
2. En esta imagen se muestra el color que se ha seleccionado para calibrar que está asociado al índice "Color 1", para obtener una buena calibración se debe seleccionar una gama de tonalidades cercanas al color deseado.
3. En la tercera imagen se muestra el resultado de aplicar el algoritmos de regionalización, se puede ver como la cámara encierra en un rectángulo la región que contiene los valores de color con los cuales fue calibrada.
4. Plano tridimensional compuesto por RGB que representa los valores de los pixeles obtenidos por la cámara, ideado por el fabricante para facilitar la selección del color que se desea calibrar y asociar tonalidades similares.

Una vez que ya se seleccionaron los valores de color que está siendo calibrado es necesario grabar esa información en la memoria de la cámara, llamada LUT (Look-Up-Table), la cual es usada durante el algoritmo de procesamiento de imágenes incluido en la HAVIMO 2.0.

3.4.1.- Algoritmo de regionalización

Su objetivo es detectar bloques continuos de color en la imagen. Usa una vecindad de 4-pixeles para determinar la conexión entre pixeles. Básicamente lo que hace es comparar los valores que encuentra en el pixel con los valores que tiene almacenados en la memoria (LUT) para determinar si pertenece al color que se calibro o no, y de esta manera formar la región. Las regiones formadas por este algoritmos siempre son rectángulos, la cámara entrega los resultados en forma de registros como se muestra en la figura 40, a cuyos valores se puede tener acceso mediante el programa RoboPlus Task, la información que puede ser consultada es el número de pixeles que forman la región y la posición del centro de la región en X y en Y, esto se obtiene al utilizar los valores de MaxX y MinX y MaxY y MinY para calcular el valor del centro.

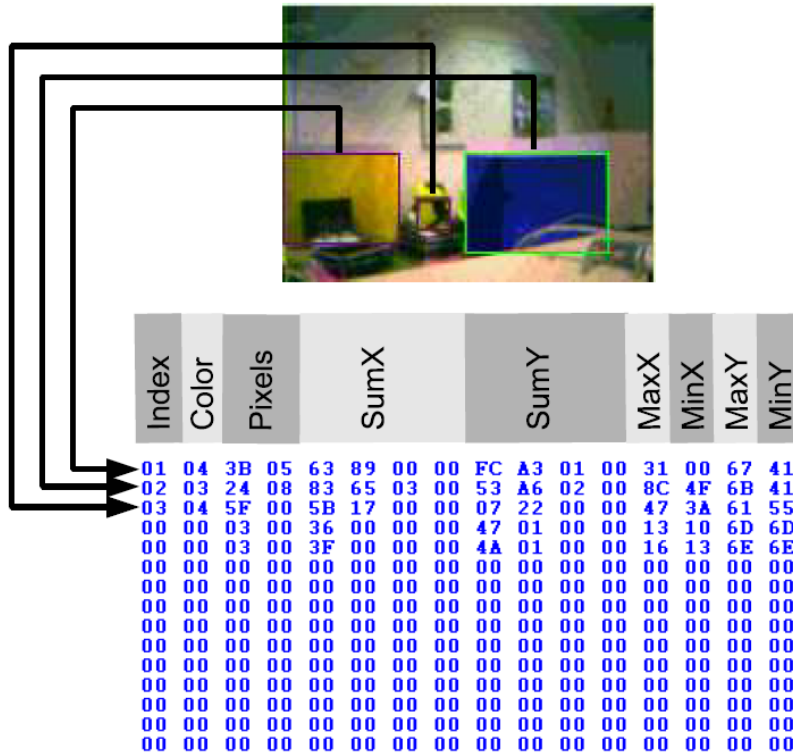


Figura 40.- Ejemplo del algoritmo de detección de color de la cámara HAVIMO.

3.5.-Sensor infrarrojo de distancia

Los sensores infrarrojos usados para este trabajo son de la marca Sharp, tienen un rango de operación de 20 cm a 150 cm, requieren de una alimentación de entre 0.3 a 7 Volts con una salida analógica de entre 0 a 3 Volts la cual no necesita otro procesamiento así que puede ser conectado directamente a un convertidor analógico a digital de un microcontrolador para obtener una medición.



Figura 41.- Sensor infrarrojo.

La curva de operación del sensor es la siguiente:

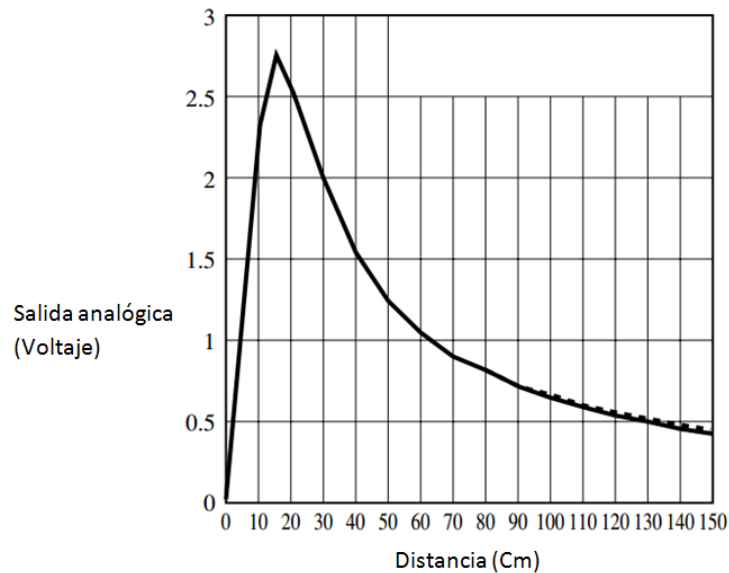


Figura 42.- Curva de operación del sensor infrarrojo.

Capítulo 4

Desarrollo

En este capítulo se explica de manera detallada todo el proceso que se realizó para desarrollar e implementar el modelo híbrido en la plataforma de pruebas. Se comienza con el ensamblado del robot y el montaje de los sensores y como fueron utilizados, después se define el algoritmo para encontrar el objetivo, se explica la implementación del sistema difuso y finalmente se presenta la red de Petri del sistema, de la cual se explican sus propiedades. También se muestra la interfaz desarrollada para controlar al robot y su funcionamiento.

4.1.-Ensamblado del robot Bioloid

El robot Bioloid se ensambla en la configuración de humanoide tipo A que cuenta con un mayor número de actuadores en las piernas y cadera con la finalidad de tener una mejor estabilidad al desplazarse. En esta configuración se requiere de 18 motores Dynamixel, pero se le adicióno uno más en la cabeza con la finalidad de montar la cámara sobre este motor para realizar el movimiento de paneo para la búsqueda de objetos.

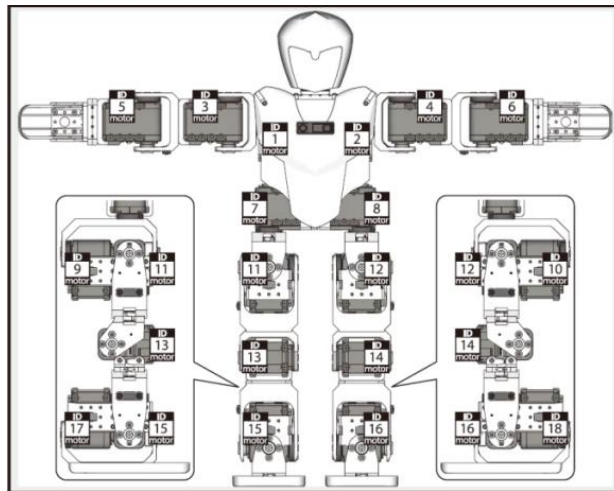


Figura 43.- Diagrama de la posición de los motores en el robot Bioloid.



Figura 44.- Robot Bioloid.

4.2.-Montaje de la cámara y los sensores

Una vez que estuvo ensamblado el robot Bioloid lo siguiente fue construir unas pequeñas piezas que ayudaran a la sujeción de la cámara y los sensores. Se selecciono el material Sintra que es espuma de cloruro de polivinilo (PVC) debido a sus propiedades de rigidez y dureza, además de que es fácil de trabajar e incluso es flexible.

Para la construcción solo fue necesario cortar algunos pedazos del tamaño que se necesitaba y hacerles unos barrenos para fijar la cámara y los sensores con algunos tornillos.



Figura 45.- Montaje de la cámara.

La cámara se montó de tal manera que dadas las características de movimiento del motor Dynamixel, que cuenta con un movimiento de 300° , permitiera obtener la mayor cantidad de información del entorno de la manera más sencilla, es decir que no se requirieran de muchos movimientos del actuador o de varios actuadores para alcanzar un área de búsqueda de 360° . Por ello se colocó el motor como se muestra en la figura 46 lo cual junto con el ángulo de visión de la cámara permitió adquirir información del entorno en un solo barrido de casi 360° .



Figura 46.- Giro del motor en el cual se montó la cámara.

En la imagen inferior se puede ver el movimiento de la cámara al realizar un paneo.



Figura 47.- Movimiento de paneo de la cámara.

Los sensores se montaron de tal forma que no interfirieran con los movimientos o con los cables del robot, también evitando que durante los movimientos realizados por el robot Bioloid los sensores se movieran generando una medición errónea, por esto se les pusieron detrás de los hombros, en esta posición pueden girar junto con los brazos pero siguen midiendo la misma distancia que tienen a su derecha e izquierda.

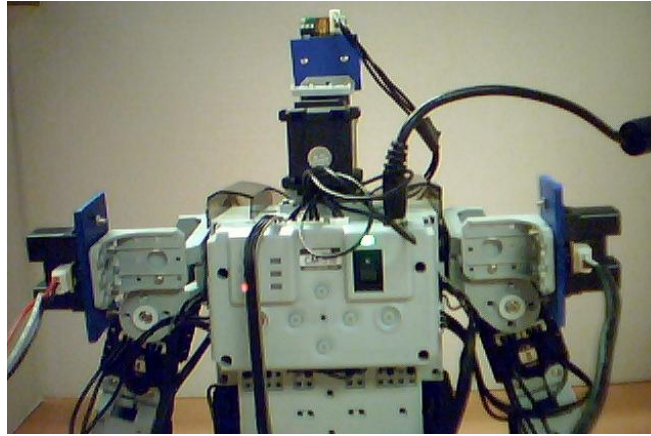


Figura 48.- Montaje de los sensores infrarrojos.

Se optó por esta configuración ya que así el robot puede obtener información de su entorno para poder buscar un camino por el cual desplazarse y así llegar al objetivo al saber hacia qué lado existe mayor espacio libre para desplazarse, hay que mencionar que esta configuración es limitada ya que al ser solo tres sensores no se conoce completamente el medio que rodea al robot lo cual puede generar errores, pero es una buena alternativa cuando se cuenta con un número limitado de sensores y una capacidad de procesamiento reducida. La información adquirida mediante estos sensores es la base para el sistema de inferencia difuso.

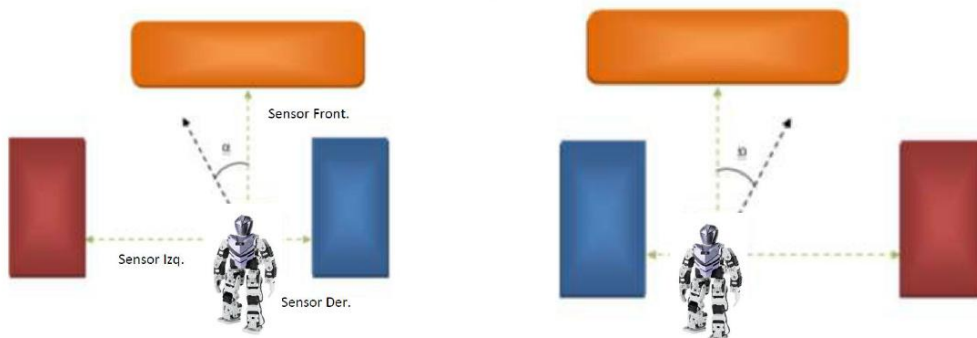


Figura 49.-Operación de los sensores infrarrojos.

4.3.-Linealización de los sensores

Los sensores ópticos utilizados en este trabajo tienen un rango de operación de entre 20 cm hasta 130 cm, que para cierta distancia entregan un valor de voltaje distinto, entre más cercano este un objeto al sensor el voltaje es mayor y va disminuyendo mientras la distancia aumenta, el cual es convertido mediante el módulo CAD del microcontrolador del robot Bioloid en un valor numérico de 8 bits, es decir que un número entre 0 y 1024. Computacionalmente estos valores son suficientes para definir el sistema difuso, pero para una persona no es representativo conocer esta información, resulta de mayor interés conocer la distancia en centímetros. Por esta razón fue necesario encontrar una función mediante la cual se pudiera pasar de los valores entregados por el convertidor analógico digital a una cantidad en centímetros.

Para llevar a cabo esto se tomo una serie de mediciones de los sensores a un objeto colocado a distintas distancias, tras lo cual se vio que los sensores laterales entregaban valores distintos al sensor frontal, debido a que estos sensores fueron adquiridos de manera separada al kit Bioloid Premium. Este cambio en los datos de los sensores no causo ningún problema salvo la necesidad de encontrar dos ecuaciones que describieran el comportamiento de los sensores.

Se tomaron alrededor de 50 muestras para distintas distancias los cuales se pueden ver en las tablas 2 y 3, con estos valores lo siguiente fue graficarlos y mediante la herramienta de ajuste de curvas del programa MATLAB fue como se encontró la ecuación que describe su comportamiento. Las ecuaciones encontradas son calculadas mediante métodos numéricos y son aproximaciones, para la finalidad de este trabajo son adecuadas.

“Navegación de un robot humanoide mediante redes de Petri y lógica difusa”

Tabla 2.- Valores de distancia medidos del sensor frontal.

Distancia (Cm)	Medición Sensor
10	552
12	468
14	395
16	345
18	305
20	275
22	255
24	235
26	218
28	202
30	191
32	179
34	165
36	157
38	149
40	140
42	136
44	129
46	125
48	120
50	116
52	111
54	107
56	103
58	99
60	95
62	91
64	89
66	87
68	83
70	82
72	78
74	75
76	73
78	71
80	69
82	67
84	65
86	65
88	63
90	62
92	57
94	57
96	57
98	54
100	54
110	50
120	45
130	41

Tabla 3.- Valores de distancia medidos de los sensores laterales.

Distancia (Cm)	Medición Sensor
15	585
17	577
19	563
21	541
23	519
25	498
27	475
29	452
31	430
33	410
35	387
37	369
39	352
41	330
43	317
45	300
47	288
49	277
51	265
53	256
55	245
57	237
59	227
61	220
63	216
65	208
67	200
69	197
71	189
73	184
75	177
77	175
79	171
81	167
83	162
85	159
87	155
89	150
91	147
93	145
95	142
97	138
99	135
101	133
110	123
115	119
120	114
125	106
130	100

Se graficaron los datos del sensor frontal de la siguiente manera. Se tomo como el eje X de la gráfica a los valores obtenidos de la medición del sensor y como el eje Y a la distancia en centímetros.

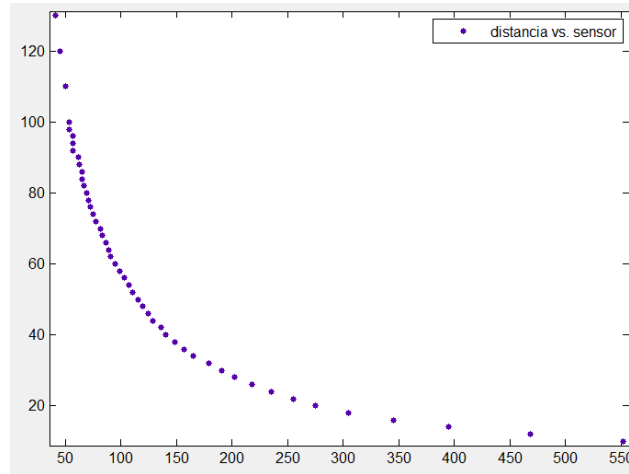


Figura 50.- Gráfica de las mediciones del sensor frontal (medición del sensor Vs. distancia).

Aplicando los métodos numéricos de aproximación de curvas se obtiene lo siguiente:

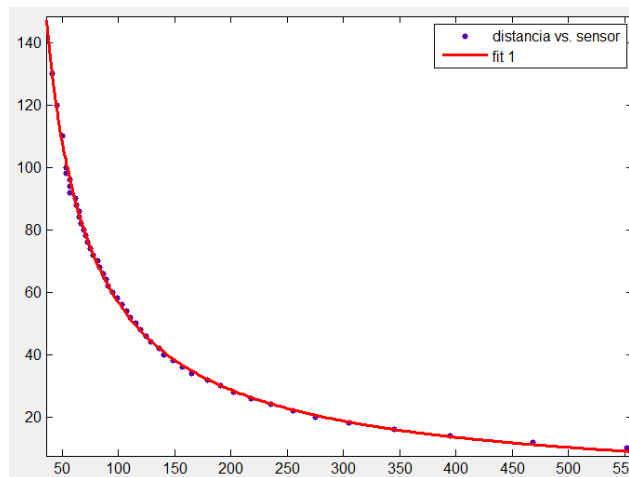


Figura 51.- Ajuste de curva del sensor frontal.

La función que se aproximó fue $y = a * x^b + c$ la cual tiene un error cuadrático medio de 0.856 y los valores obtenidos para los coeficientes fueron:

$$a = 3579 \quad b = -0.8831 \quad c = -4.526$$

$$y = 3579 * x^{-0.8831} - 4.526 \quad \text{Ecuación para linealizar el sensor frontal}$$

Se hizo el mismo procedimiento para los sensores laterales, y el resultado fue el siguiente:

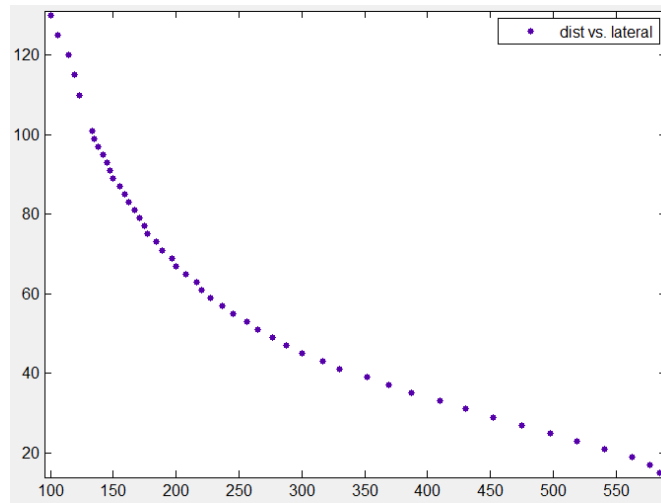


Figura 52.- Gráfica de las mediciones de los sensores laterales (medición del sensor Vs. distancia).

Aplicando los métodos numéricos de aproximación de curvas se obtiene:

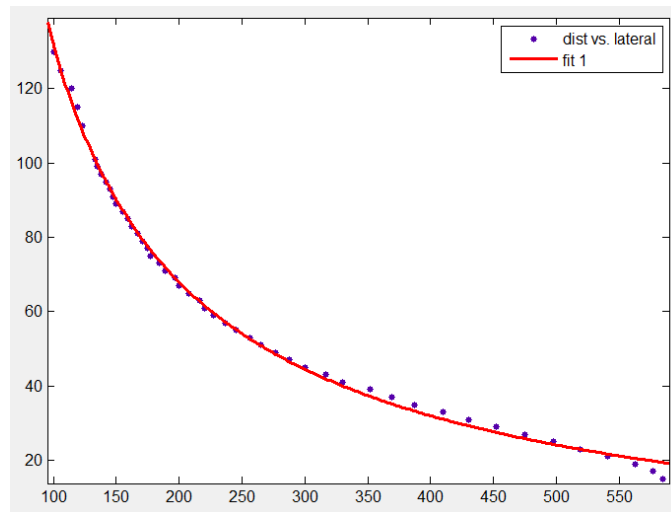


Figura 53.- Ajuste de curva de los sensores laterales.

La función que se aproximó fue $y = a * x^b + c$ la cual tiene un error cuadrático medio de 1.384 y los valores obtenidos para los coeficientes fueron:

$$a = 6500 \quad b = -0.8235 \quad c = -14.8$$

$$y = 6500 * x^{-0.8235} - 14.8 \quad \text{Ecuación para linealizar los sensores laterales}$$

4.4.-Calibración de la cámara

Para obtener buenos resultados al trabajar con la cámara HAVIMO 2.0 es primordial haber hecho una buena calibración, esto debido a cómo funciona el algoritmo de regionalización de colores en la imagen, ya que es muy susceptible al ruido producido en la imagen por la luz o la sombra del medio, lo cual repercute en el funcionamiento de la cámara ya que si se calibra solo para una condición de iluminación es probable que al cambiar ligeramente estas condiciones la cámara ya no sea capaz de detectar la región deseada, para evitar esto es recomendable hacer una calibración del color desde distintas distancias y desde distintas posiciones para de esta manera asociar a un solo color toda la gama de tonalidades que pueden presentarse debido a la iluminación y de esta forma obtener mejores resultados en los experimentos. El proceso de calibración de la cámara para este trabajo fue el que se describe a continuación.

Lo primero fue seleccionar el color que se deseaba buscar, en este caso se optó por el color verde debido a las características del medio en el cual se probó el robot durante los experimentos ya que la superficie del piso donde caminó el robot fue de color anaranjado o amarillo y las paredes que se construyeron fueron de color café y blanco, entonces el color verde resulta contrastante o distinto a todos los demás presentes en el medio, con el objetivo de evitar ruido o que el robot detecte regiones no deseadas debido a los efectos de la iluminación.

Para la calibración se colocó al robot en distintos lugares del área de trabajo junto con el objetivo que se buscó durante las pruebas, se mantuvo el objetivo en una sola posición y se cambió de lugar al robot como se muestra en la figura 54.

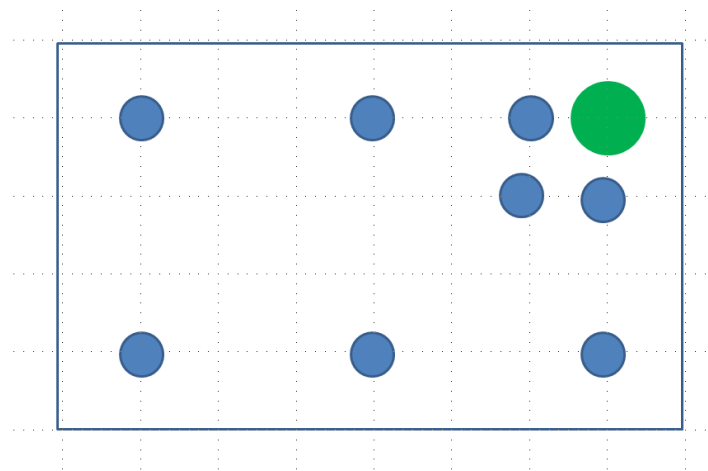


Figura 54.- Posiciones en las cuales se realizó la calibración.

(la esfera verde es el objetivo y el azul es el robot)

Las imágenes tomadas mediante la interfaz GUI de la cámara para calibrar se muestran a continuación en la figura 55, donde se ve que las tonalidades seleccionadas para pertenecer al color que se está calibrando son sustituidas por píxeles de color turquesa, en la parte inferior se puede ver la gráfica tridimensional de RGB con los píxeles elegidos como el color que buscara el robot bioloid.

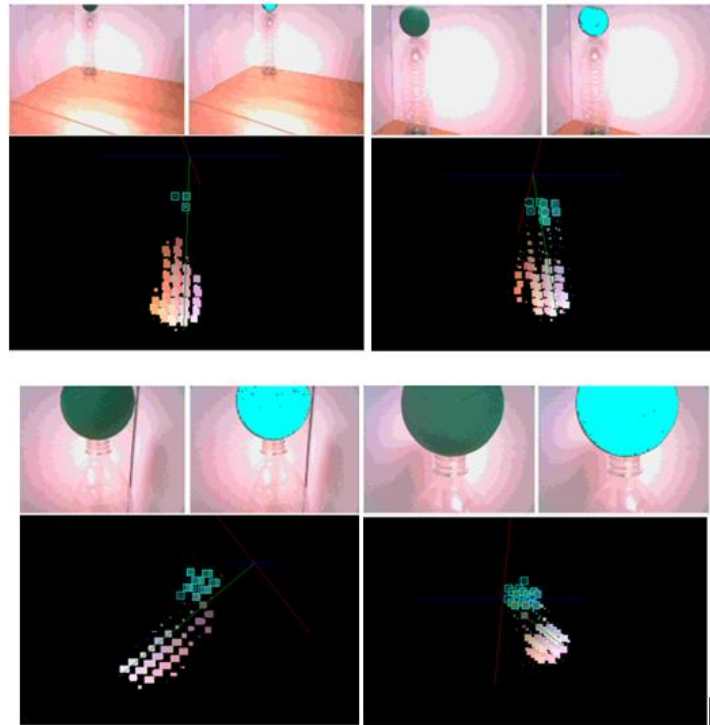


Figura 55.- Imágenes que muestran la calibración.

En las imágenes se puede ver cómo debido a las diferentes distancias es necesario seleccionar una mayor cantidad de tonalidades del color del objetivo, en color turquesa se puede ver la región que el algoritmo de la cámara aislara en la imagen y de la cual nos dará la información del número de píxeles que la conforma y el centro de la región.



Figura 56.- Funcionamiento del algoritmo de detección de color de la cámara HAVIMO.

4.5.-Cálculo de distancia mediante la cámara

Para colocar en el área de trabajo el objeto que sería buscado por la cámara se utilizó una estructura con una altura de 33 centímetros para que el robot pudiera posicionarse frente a él, esto se hizo con la idea de que el robot pudiera acercarse al objeto con la información obtenida de la cámara al determinar a qué distancia se encuentra del objeto y entonces desplazarse un número de pasos adecuado para no derribar el objetivo y posicionarse a una distancia correcta para finalizar la tarea. Al usar este método se obtuvieron buenos resultados ya que se seleccionó que el objeto que se buscaría fuera una esfera de 8 centímetros de diámetro, como ya se mencionó de color verde, con el objetivo de que el robot mediante la cámara fuera capaz de obtener la misma imagen del objeto sin importar desde qué punto lo observara, es decir evitar transformaciones de perspectiva en la imagen.

Para poder determinar la distancia usando la cámara, mediante el número de píxeles de la región del color buscado, lo que se hizo fue tomar una serie de imágenes del objeto colocado a distintas distancias de la cámara y se obtuvo el número de píxeles para cada caso. Debido a que el diámetro del objeto no cambia de acuerdo a la distancia que se encuentre de la cámara, sino que el cambio ocurre en el número de píxeles que representa su superficie, por esta razón es posible hallar una relación entre el número de píxeles encontrados y la distancia a la cual se localiza el objeto. Lo siguiente fue graficar los resultados obtenidos en dichas mediciones y usando la herramienta de ajuste de curvas de MATLAB encontrar la función que describe su comportamiento.

Tabla 4.- Medida del número de píxeles del objetivo a distintas distancias

Distancia (Cm)	Número de píxeles
10	10000
20	3800
30	1800
50	650
70	320
90	190
100	160

En la gráfica podemos ver que el eje X representa el número de píxeles y el eje Y representa la distancia en centímetros a la cual se encuentra el objeto de la cámara.

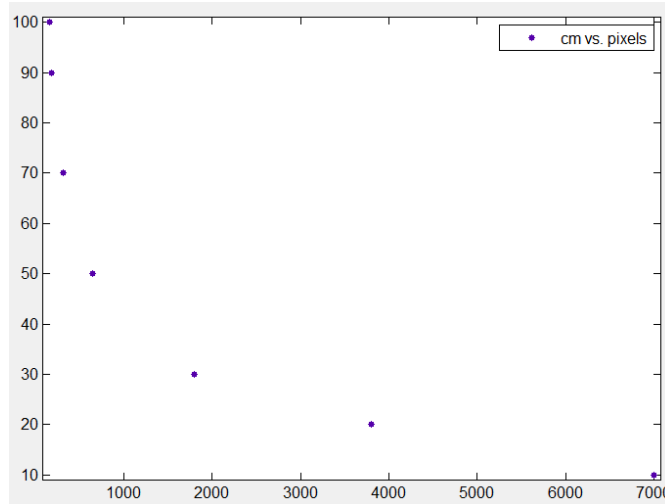


Figura 57.- Gráfica de número de píxeles Vs. distancia al objetivo.

Aplicando los métodos numéricos de aproximación de curvas se obtiene:

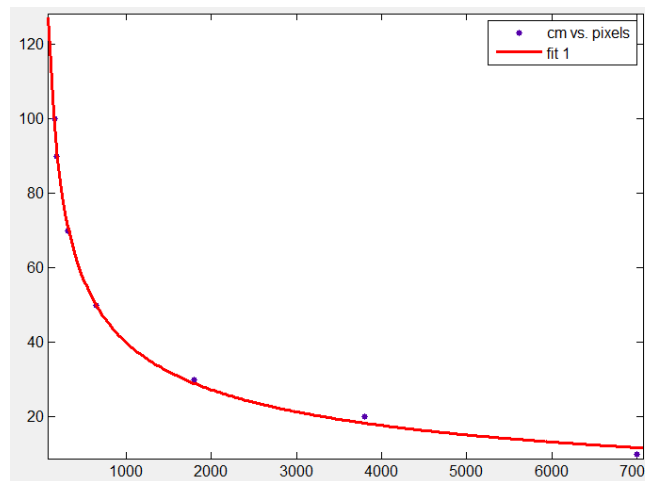


Figura 58.- Ajuste de curva para calcular la distancia mediante la cámara HAVIMO.

La función que se aproximó fue $y = a * x^b + c$ la cual tiene un error cuadrático medio de 1.384 y los valores obtenidos para los coeficientes fueron:

$$a = 905.5 \quad b = -0.4162 \quad c = -10.96$$

$$y = 905.5 * x^{-0.4162} - 10.96$$

Ecuación para calcular la distancia al objetivo.

4.6.-Movimientos del robot para caminar y girar

El movimiento de un robot humanoide es un problema complejo que se ha atacado desde distintas perspectivas, como se menciona en la introducción sigue siendo una área de investigación recurrente, este trabajo está enfocado a la navegación de un robot humanoide para el cual es necesario que el robot tenga un desplazamiento estable y pueda moverse y realizar giros dentro del área elegida para las pruebas, dada la compleja tarea que sería controlar el giro de cada motor y sintonizar el ritmo de un caminado estable y que este trabajo no está enfocado a realizar un análisis de marcha para un robot humanoide es que se opto por utilizar los movimiento que brinda el fabricante para que el robot bioloid camine y gire los cuales vienen incluidos en un archivo de prueba del programa RoboPlus Motion.

Este programa incluye los siguientes movimientos:

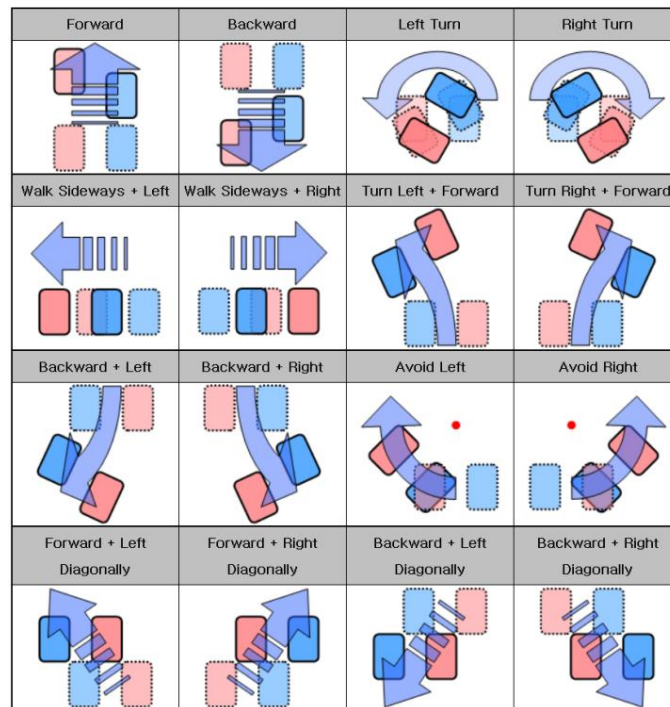


Figura 59.- Movimientos proporcionados por el fabricante que puede realizar el robot.

Debido a que estos movimientos aunque le permiten al robot bioloid desplazarse no son del todo estables, ya que aún están sometidos a otros factores que los alteran como son el desgaste en los engranes de los motores, la fricción que exista entre la superficie donde

caminara el robot y sus pies, alteraciones en la superficie donde se desplace el robot, es decir pequeños desniveles, inclinaciones o rugosidades que pueden ocasionar que el robot no camine bien o incluso que se caiga, entre otros factores externos que afectan el caminado del robot bioloid.

A pesar de las desventajas que tienen estos movimientos se decidió utilizarlos para este trabajo, pero no se utilizaron todos los movimientos contenidos en el archivo de ejemplo, solo se tomaron los giros a la derecha y a la izquierda y el caminado hacia adelante. No se utilizo directamente el mismo archivo de los ejemplos sino que se modifico y se adapto a las necesidades de este trabajo. Los giros del robot son aproximadamente de 25° a 30° y se ajustaron para que pudieran iterarse y que el robot pudiera realizar varios giros seguidos de ser necesarios.

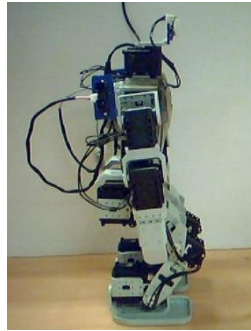


Figura 60.- Postura del robot al caminar.

Con respecto al caminado del robot se realizaron tres rutinas diferentes con el objetivo de elegir cual realizar debido a las condiciones del entorno.

- Caminado Corto: El robot bioloid solo da un paso, esto es para que pueda acercarse al objetivo sin derribarlo.
- Caminado Medio: El robot bioloid da tres pasos.
- Caminado Largo: El robot bioloid da seis pasos para usarse cuando necesita avanzar distancias grandes.

También se implemento un movimiento para que fuese realizado después de haber hallado el objeto, la tarea que realiza el robot una vez que esta a una distancia adecuada frente al objetivo es derribar la esfera verde. Para realizar esto fue necesario crear un movimiento con el brazo derecho del robot el cual le diera un golpe a la esfera para derribarla, se utilizo el programa RoboPlus Motion donde mediante una secuencia de movimientos de los motores que conforman el brazo derecho fue generada esta tarea.

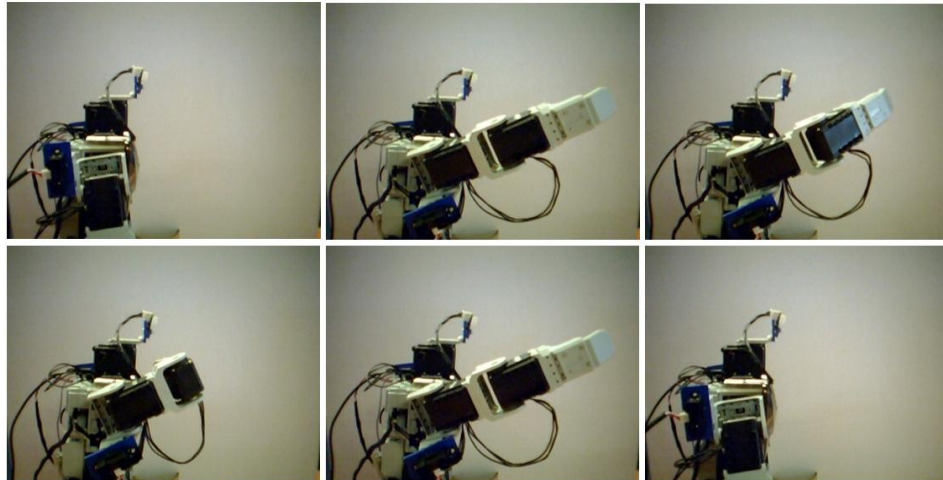


Figura 61.- Movimiento para derribar el objetivo.

4.7.-Algoritmo para ubicar el objeto y posicionar el robot.

Una vez que se terminó de ensamblar, de montar la cámara y que se implementaron los movimientos necesarios para que el robot girara y caminara, además de la calibración del color que se buscaría lo siguiente que se realizó fue desarrollar un algoritmo mediante el cual el robot bioloid fuera capaz de localizar el objeto en un espacio libre de obstáculos alinearse al objeto y caminar hacia él para derribarlo con el brazo derecho.

Para desarrollar este algoritmo primero se definió como realizaría la búsqueda el robot mediante la cámara, por ello se propuso que el robot realizará un paneo el cual consiste en colocar el motor que mueve a la cámara en 9 posiciones distintas. La tabla de abajo explica las posiciones del motor en base a como se montó la cámara, ver figura 46.

Tabla 5.- Posiciones del movimiento de paneo.

Regiones	Posición en grados	Valor de la posición del motor
1	-150°	1024
2	-112.5°	896
3	-75°	768
4	-37.5°	640
5	0°	512
6	37.5°	384
7	75°	256
8	112.5°	128
9	150°	0

Una vez que la cámara ha localizado el objetivo, si no es encontrado la cámara regresa a su posición inicial es decir 0°, la siguiente tarea que realiza el robot bioloid es girar un número de veces para tratar de posicionarse frente al objetivo, esto se hace mediante la posición del motor que sostiene la cámara, ya que el valor de la posición del motor está entre 0 y 1023 entonces mediante la siguiente formula se puede calcular el número de veces que debe girar a la derecha o izquierda para colocarse frente al objeto, la siguiente formula explica lo anterior:

$$Gn = \frac{Po - Pi}{Gr}$$

Donde:

- *Gn*.- Es el número de giros que debe realizar el robot, su valor es de -4 a 4.
- *Po*.- Es la posición inicial de la cámara, es siempre de 512.
- *Pi*.- Es la posición actual de la cámara, sus valores pueden verse en la *tabla*.
- *Gr*.- Es el número de grados que puede girar el robot en un movimiento, es de entre 25° a 30° se toma como 80 al hacer la conversión a valores de posición del motor.

De la ecuación anterior podemos ver que cuando el signo es negativo se trata de un giro a la izquierda, el número que toma el robot es solo el valor entero que se calcula los decimales son omitidos. En el mejor de los casos el robot después de realizar los giros se localizara frente al objeto, pero debido a la imprecisión en sus movimiento esto no ocurre siempre por lo tanto se requiere de un posicionamiento más preciso del robot antes de que comience a acercarse al objetivo.

Para ello ahora que el robot ya está frente al objetivo lo que trata de hacer es que el centro de la región se encuentre en la posición central de la imagen, para este caso solo nos interesa el valor de la coordenada X, se estableció una zona en la cual se busca que se localice el centro de la región, ver figura 62, para poder lógralo dependiendo del valor de la coordenada X el motor que sostiene la cámara girara 2.5° a la izquierda o a la derecha para hacer que el centro se encuentre en la zona deseada. Si el número de giros hechos a la izquierda o a la derecha son mayores que 30° esto quiere decir que el robot necesita girar nuevamente para posicionarse frente al objetivo por lo tanto realiza el giro y el proceso de posicionar el centro en la zona deseada de la imagen se repite, si el valor del giro es menor a 30° se considera que esta alineado y el robot puede comenzar a caminar después de recibir la instrucción de la computadora.

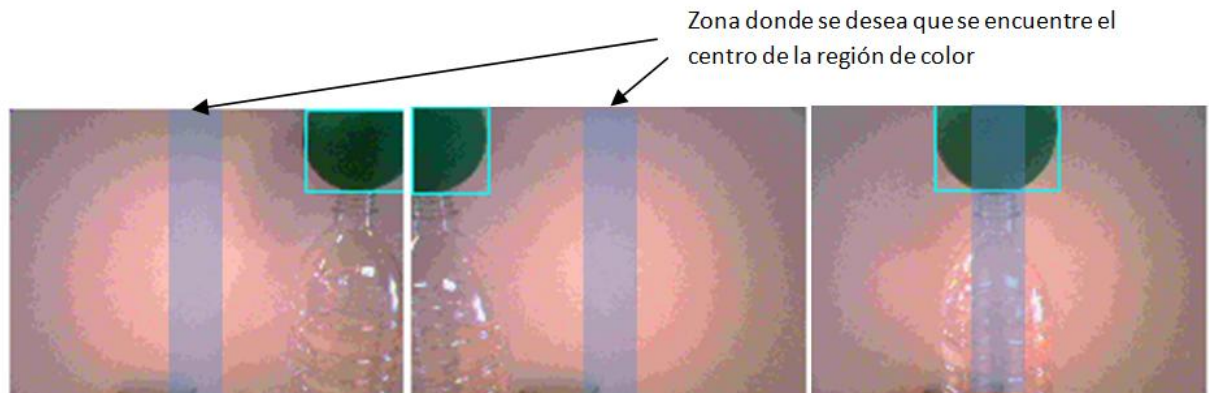


Figura 62.- Zona donde debe encontrarse el centro del objetivo.

Una vez que el robot se ha posicionado frente al objeto de manera correcta, es decir que el objetivo se encuentra centrado, entonces el robot está listo para comenzar a acercarse, para ello envía la información del objetivo a la computadora para que calcule a qué distancia se encuentra y dependiendo del valor encontrado el robot realizará un caminado mediano si la distancia es mayor a 30 cm de lo contrario realizará un caminado corto hasta que logre posicionarse a entre 8 y 10 cm del objeto para finalizar su tarea y derribar la pelota verde. Todo el proceso de este algoritmo fue programado en el micro controlador del robot *bioloid*, menos la ecuación para calcular la distancia con la cámara, así que la búsqueda los giros y el centrado mediante la cámara lo realiza sin necesidad de una instrucción de la computadora, pero caminará hasta que la computadora haya calculado la distancia y le indique que movimiento realizar.

4.8.-Sistema difuso de toma de decisión (Sistema difuso de inferencia)

El sistema difuso de toma de decisión es el encargado de la navegación del robot mientras el objeto no haya sido encontrado, mediante los valores de los sensores que le brindan información del entorno debe ser capaz de desplazarse en el ambiente evitando los obstáculos, siempre moviéndose hacia adelante y hacia donde encuentre un mayor espacio libre para realizar una exploración con la intención de encontrar el objeto.

El tipo de sistema que se utilizó es un sistema difuso de inferencia Mamdani con tres entradas, las cuales son las distancias medidas por los sensores frontal, izquierdo y derecho que se obtienen mediante sus respectivas ecuaciones de linealización. Cuenta con un conjunto de reglas que le permiten al *robot bioloid* desplazarse como un robot

seguidor de paredes. Después de evaluar las reglas y de calcular el valor de salida mediante la defusificación lo que se obtiene es el siguiente movimiento que realizará el robot de acuerdo a las condiciones en que se encuentra, los cuales pueden ser girar a la derecha o izquierda, realizar un giro de 180° o un caminado largo, medio o corto.

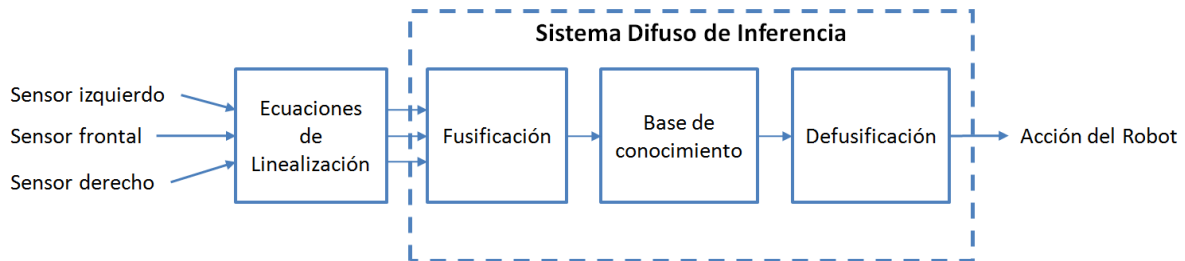


Figura 63.- Diagrama del funcionamiento del sistema difuso de inferencia.

4.8.1.- Conjuntos difusos de entrada

Como ya se menciona las entradas del sistema difuso son los sensores, para los cuales se definió un universo de discurso y funciones de membresía asociadas a conceptos difusos como "Cerca", "Medio" y "Lejos". En este caso debido a las características de los sensores se definieron distintas funciones de membresía para el sensor frontal y para los sensores laterales, para los sensores laterales se utilizaron las mismas funciones de membresía ya que su comportamiento es igual y se desea mapear los mismos conceptos difusos.

4.8.1.1.- Conjuntos difusos de los sensores laterales

Para los sensores frontales se definieron tres funciones de membresía asociadas a los conceptos difusos siguientes:

- Cerca: Se refiere a que existe un obstáculo cercano a la izquierda o la derecha del robot, se representa por una función sigmoide de membresía.
- Medio: Se refiere a que existe un obstáculo a la izquierda o la derecha del robot, se representa por una función campana de Gauss de membresía.
- Lejos: Se refiere a que existe un obstáculo lejano a la izquierda o la derecha del robot, se representa por una función sigmoide de membresía.

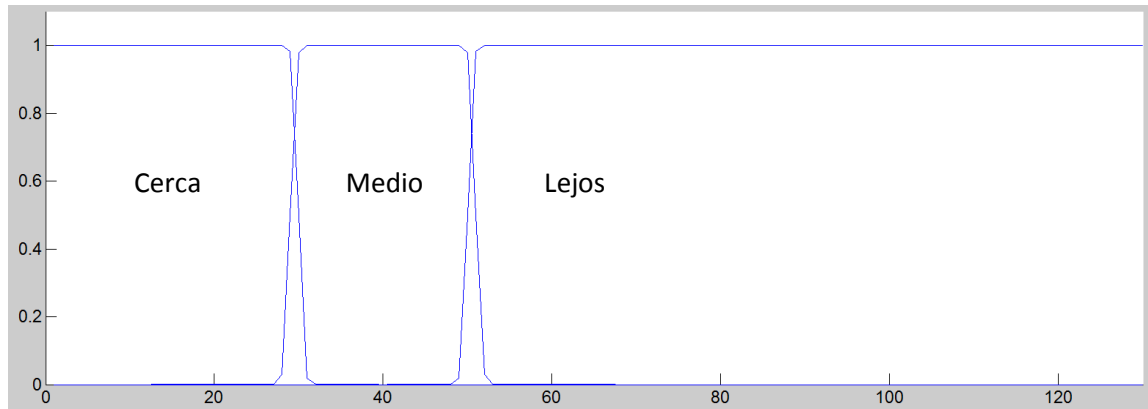


Figura 64.- Conjuntos difusos de los sensores laterales.

La figura anterior muestra la representación gráfica de las funciones de membresía para los sensores laterales, los valores de los coeficientes de las funciones fueron elegidos de manera intuitiva y sus valores se muestran a continuación:

Tabla 6.- Coeficientes de las funciones de membresía de los sensores laterales.

Concepto difuso	Coeficientes			Ecuación de la función de membresía
	a	b	c	
Cerca	-4	---	30	$f(x, a, c) = \frac{1}{1 + e^{-a(x-c)}}$
Medio	11	20	40	$f(x, a, b, c) = \frac{1}{1 + \left \frac{x-c}{a} \right ^{2b}}$
Lejos	4	---	50	$f(x, a, c) = \frac{1}{1 + e^{-a(x-c)}}$

4.8.1.2.- Conjuntos difusos del sensor frontal

Para el sensor frontal se definieron cuatro funciones de membresía asociadas a los conceptos difusos siguientes:

- Cerca: Se refiere a que existe un obstáculo cercano al frente del robot, se representa por una función sigmoide de membresía.
- Medio cerca: Se refiere a que existe un obstáculo moderadamente cerca al frente del robot, se representa por una función campana de Gauss de membresía.

- Medio: Se refiere a que existe un obstáculo a una distancia media al frente del robot, se representa por una función campana de Gauss de membresía.
- Lejos: Se refiere a que existe un obstáculo lejano al frente del robot, se representa por una función sigmoide de membresía.

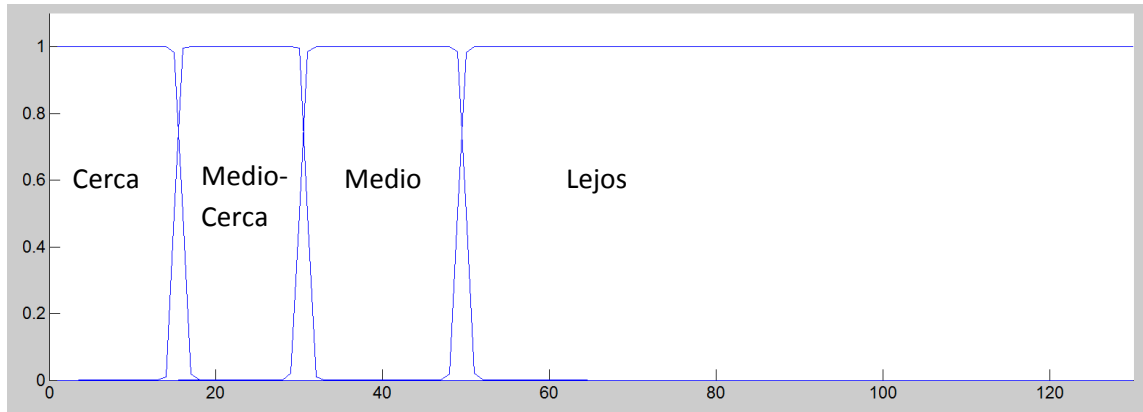


Figura 65.- Conjuntos difusos del sensor frontal.

La figura anterior muestra la representación gráfica de las funciones de membresía para el sensor frontal, los valores de los coeficientes de las funciones fueron elegidos de manera intuitiva y sus valores se muestran a continuación:

Tabla 7.- Coeficientes de las funciones de membresía del sensor frontal.

Concepto difuso	Coeficientes			Ecuación de la función de membresía
	a	b	c	
Cerca	-4	---	16	$f(x, a, c) = \frac{1}{1 + e^{-a(x-c)}}$
Medio - Cerca	8	20	23	$f(x, a, b, c) = \frac{1}{1 + \left \frac{x-c}{a} \right ^{2b}}$
Medio	10	20	40	$f(x, a, b, c) = \frac{1}{1 + \left \frac{x-c}{a} \right ^{2b}}$
Lejos	4	---	49	$f(x, a, c) = \frac{1}{1 + e^{-a(x-c)}}$

Con estos valores de sus coeficientes se hicieron las primeras pruebas con el robot pero los resultados no fueron los esperados por ello fue necesario modificarlos para tener un

mejor comportamiento del sistema difuso. El ajuste de los coeficientes se hizo de manera manual y durante la fase de experimentos, los valores que se eligieron y que funcionaron mejor fueron los siguientes

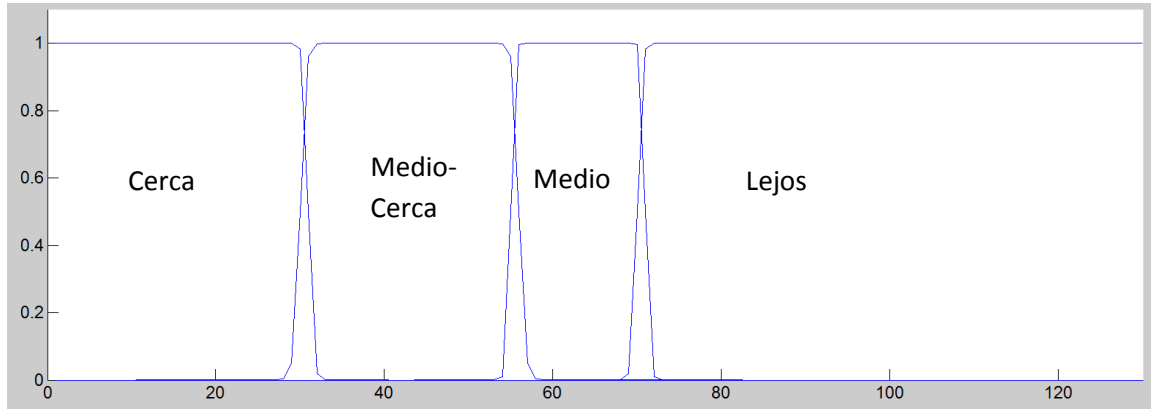


Figura 66.- Conjuntos difusos finales del sensor frontal.

Tabla 8.- Coeficientes finales de las funciones de membresía del sensor frontal.

Concepto difuso	Coeficientes			Ecuación de la función de membresía
	<i>a</i>	<i>b</i>	<i>c</i>	
Cerca	-4	---	31	$f(x, a, c) = \frac{1}{1 + e^{-a(x-c)}}$
Medio - Cerca	13	20	43	$f(x, a, b, c) = \frac{1}{1 + \left \frac{x-c}{a} \right ^{2b}}$
Medio	8	20	63	$f(x, a, b, c) = \frac{1}{1 + \left \frac{x-c}{a} \right ^{2b}}$
Lejos	4	---	70	$f(x, a, c) = \frac{1}{1 + e^{-a(x-c)}}$

Se eligieron estos coeficientes porque el coeficiente *c* de las funciones de membresía es el umbral que determina cuando una distancia pertenece en mayor grado a uno u otro conjunto. En la experimentación se vio que el robot debe mantenerse a cierta distancia del obstáculo para poder realizar sus giros y evitarlo, esto debido a las ya antes mencionadas fallas o errores generados por los propios movimientos del robot, por esta razón y por cómo fueron definidas las reglas difusas se eligieron estos coeficientes para dar una tolerancia al error generado por los movimientos del robot con el objetivo de que realice su tarea de manera satisfactoria al buscar evitar las colisiones con las paredes al tratar de mantenerse siempre a una distancia segura, en este caso a partir de que se

encuentre a 30 cm o menos del obstáculo el robot intentara evitarlo de acuerdo a como se explica en las reglas.

4.8.2.- Conjunto difuso de salida

Lo que se eligió tener como salida del sistema difuso de inferencia fue un movimiento o una acción del robot, esto debido a que no se está controlando una variable en específico sino que se busca que el robot pueda actuar de acuerdo a la información de sus sensores y tome la mejor decisión basado en las reglas para seguir con la exploración. Para esto se definieron siete acciones: caminado largo, caminado medio, caminado corto, giro a la izquierda, giro a la derecha, giro de 180° y caso especial, las cuales son las salidas del sistema difuso.

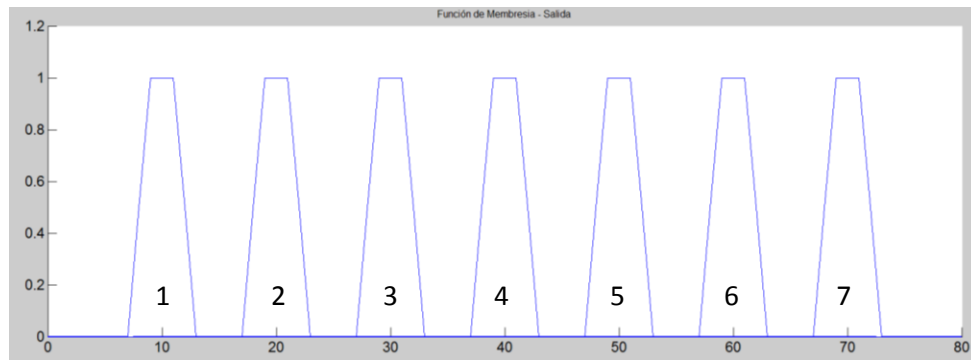


Figura 67.- Conjuntos difusos de salida.

Tabla 9.- Coeficientes de las funciones de membresía de salida.

Función de membresía	Concepto difuso	Coeficientes			Ecuación de la función de membresía
		<i>a</i>	<i>b</i>	<i>c</i>	
1	Caminado largo	2	10	10	$f(x, a, b, c) = \frac{1}{1 + \left \frac{x - c}{a} \right ^{2b}}$
2	Caminado medio	2	10	20	
3	Caminado corto	2	10	30	
4	Giro a la izquierda	2	10	40	
5	Giro a la derecha	2	10	50	
6	Giro de 180°	2	10	60	
7	Caso especial	2	10	70	

A la salida del sistema se le aplica el método de defusificación por máximo medio, se realizó esto para poder tener un valor cercano a los centros de las superficies difusas que son los valores que nos interesa enviar al robot para que el pueda identificar de manera más sencilla que tarea es la que debe realizar.

5.8.3.- Definición de las reglas difusas

Debido a que el sistema difuso cuenta con 3 entradas, de las cuales una está definida por 4 funciones de membresía y las otras dos entradas están definidas por 3 conjuntos difusos el número de reglas que puede formarse es de 36. Aquí es donde se puede observar una de las desventajas de los sistemas difusos ya que si hubiera más entradas o si se hubiera definido un número mayor de conjuntos difusos para cada entrada el número de reglas que podrían formarse crecería de manera exponencial, simplemente con una entrada mas con tres funciones de membresía el número de reglas que se podrían formar sería de 108, o bien si se hubieran definido 4 conjuntos difusos para cada una de las tres entradas tendríamos 64 reglas posibles.

Con el objetivo de tener un sistema robusto pero mantener un número de reglas bajo es que se decidió utilizar el número de funciones de membresía que ya se explico. El comportamiento o el razonamiento que se busca emular con este sistema difuso es el de un robot seguidor de paredes, cuyo funcionamiento consiste en desplazarse hacia adelante hasta encontrarse a una distancia segura de la pared u obstáculo para entonces realizar un giro, a la derecha o izquierda, dependiendo de hacia dónde encuentre un mayor espacio libre. Además se incluye la posibilidad de que el robot pueda realizar un giro de 180° en caso de que encuentre un camino sin salida durante su exploración.

Como ya se dijo se tiene un total de 36 reglas para este sistema debido a como se fusifican las 3 entradas, son muchas reglas para definir las una a una, así que lo que se hizo fue proponer dos reglas que pudieran englobar a algunas otras, estas reglas fueron:

- ❖ **SI** hay un obstáculo lejos al frente **entonces** realizar un caminado largo.

- ❖ **SI** hay un obstáculo a una distancia media al frente **entonces** realizar un caminado medio.

Estos enunciados están basados en la idea de darle prioridad a que el robot camine hacia adelante buscando acercarse a la pared o el obstáculo que este frente a él, por esta razón es que en las reglas anteriores no influyen los valores de los sensores izquierdo y derecho ya que cualquiera que sea su valor no influye en el cálculo del peso de estas dos reglas, en ellas se incluyen a 18 otras reglas, lo que reduce el número de reglas por definir a la mitad.

Las reglas restantes se pueden agrupar dependiendo de cuál es la acción que se desea que realice el robot con respecto a los valores de entrada del sistema. Como por ejemplo para girar 180° la regla es:

- ❖ **SI** hay un obstáculo cerca al frente **Y** hay un objeto cerca a la izquierda **Y** hay un obstáculo cerca a la derecha **entonces** realizar un giro de 180°.

Otra de las acciones que se busca que realice el robot es hacer un caminado corto para acercarse al obstáculo con la idea de mantenerse a una distancia segura, por esta razón el robot da un paso a la vez, y entonces decidir sí se realiza un giro a la derecha o la izquierda, las reglas que definen esta situación son 3 que se muestran a continuación:

- ❖ **SI** hay un obstáculo medio-cerca al frente **Y** hay un obstáculo cerca a la izquierda **Y** hay un obstáculo cerca a la derecha **entonces** realizar un caminado corto.
- ❖ **SI** hay un obstáculo medio-cerca al frente **Y** hay un obstáculo a una distancia media a la izquierda **Y** hay un obstáculo a una distancia media a la derecha **entonces** realizar un caminado corto.
- ❖ **SI** hay un obstáculo medio-cerca al frente **Y** hay un obstáculo lejos a la izquierda **Y** hay un obstáculo lejos a la derecha **entonces** realizar un caminado corto.

Para realizar un giro a la izquierda, se formaron 6 reglas que al reducirse se expresan de la siguiente forma:

- ❖ **SI** hay un obstáculo a una distancia media al frente **O** hay un obstáculo cerca al frente **Y** hay un obstáculo a una distancia media a la izquierda **Y** hay un obstáculo cerca a la derecha **entonces** realizar un giro a la izquierda.
- ❖ **SI** hay un obstáculo a una distancia media al frente **O** hay un obstáculo cerca al frente **Y** hay un obstáculo lejos a la izquierda **Y** hay un obstáculo cerca a la derecha **entonces** realizar un giro a la izquierda.

- ❖ **SI** hay un obstáculo a una distancia media al frente **O** hay un obstáculo cerca al frente **Y** hay un obstáculo lejos a la izquierda **Y** hay un obstáculo a una distancia media a la derecha **entonces** realizar un giro a la izquierda.

Para realizar un giro a la derecha las reglas también se formaron 6 reglas que son similares a las anteriores:

- ❖ **SI** hay un obstáculo a una distancia media al frente **O** hay un obstáculo cerca al frente **Y** hay un obstáculo cerca la izquierda **Y** hay un obstáculo a una distancia media a la derecha **entonces** realizar un giro a la derecha.
- ❖ **SI** hay un obstáculo a una distancia media al frente **O** hay un obstáculo cerca al frente **Y** hay un obstáculo cerca a la izquierda **Y** hay un obstáculo lejos a la derecha **entonces** realizar un giro a la derecha.
- ❖ **SI** hay un obstáculo a una distancia media al frente **O** hay un obstáculo cerca al frente **Y** hay un obstáculo a una distancia media a la izquierda **Y** hay un obstáculo lejos a la derecha **entonces** realizar un giro a la derecha.

Finalmente existen dos condiciones que se pueden considerar casos especiales, las cuales son las siguientes:

- ❖ **SI** hay un obstáculo cerca al frente **Y** hay un obstáculo a una distancia media a la izquierda **Y** hay un obstáculo a una distancia media a la derecha **entonces** es un caso especial.
- ❖ **SI** hay un obstáculo cerca al frente **Y** hay un obstáculo lejos a la izquierda **Y** hay un obstáculo lejos a la derecha **entonces** es un caso especial.

Estos casos especiales las condiciones sugieren que el robot debe realizar un giro para evitar el obstáculo, pero como hacia ambos lados tiene un espacio libre similar no se puede determinar hacia qué lado es mejor girar, por esta razón se consideran casos especiales. Para solucionar esto dentro de la interfaz desarrollada se elige que se debe realizar el giro anterior en estos casos especiales, es decir que si el robot había girado a la izquierda previamente entonces si se presenta un caso especial el robot gira a la izquierda, en cambio si el último giro que realizó el robot fue a la derecha entonces si se presenta un caso especial entonces girara a la derecha.

La siguiente tabla expresa todas las reglas del sistema difuso:

Tabla 10.- Reglas del sistema difuso.

Reglas	Sensor izquierdo	Sensor frontal	Sensor derecho	Decisión (acción)
1 - 9	N/A	Lejos	N/A	Caminado Largo
9 - 18	N/A	Media	N/A	Caminado Medio
19	Cerca	Cerca	Cerca	Giro 180°
20	Cerca	Medio-Cerca	Cerca	Caminado Corto
21	Media	Medio-Cerca	Media	Caminado Corto
22	Lejos	Medio-Cerca	Lejos	Caminado Corto
23 - 24	Media	Medio-Cerca / Cerca	Cerca	Giro a la izquierda
25 - 26	Lejos	Medio-Cerca / Cerca	Cerca	Giro a la izquierda
27 - 28	Lejos	Medio-Cerca / Cerca	Media	Giro a la izquierda
29 - 30	Cerca	Medio-Cerca / Cerca	Media	Giro a la derecha
31 - 32	Cerca	Medio-Cerca / Cerca	Lejos	Giro a la derecha
33 - 34	Media	Medio-Cerca / Cerca	Lejos	Giro a la derecha
35	Media	Cerca	Media	Caso especial
36	Lejos	Cerca	Lejos	Caso especial

Para evaluar las reglas se utilizaron los operadores MIN y MAX como se hace en los sistemas Mamdani, es decir que para calcular el conector "Y" se utiliza el operador MIN y para calcular el operador "O" se usó el operador MAX.

4.8.4.- Implementación del sistema difuso

El sistema difuso se implemento mediante un código desarrollado en MATLAB, donde se crean las funciones de membresía de entrada y salida para un conjunto de 130 elemento, que son los valores de distancia que mide el sensor, de 0 a 130 cm considerando solo valores enteros. Es decir que se tienen tres conjuntos de 130 elementos que representan cada una de las entradas, esto es la distancia medida por los sensores.

Con los valores de estos tres conjuntos se evalúan y se obtienen sus respectivos grados de pertenencia para todas las funciones de membresía, las cuales son evaluadas mediante las reglas definidas por los operadores MAX y MIN para calcular los pesos respectivos y encontrar la superficie de salida para aplicarle el método de defusificación. Finalmente se obtiene una base de conocimiento (base de datos) que está representada por tres índices, que son los tres sensores de entrada, formada por 130x130x130 elementos que guardan un valor que representa a las acciones del robot. Esto significa que el sistema de inferencia

Mamdani solo tiene que ser calculado una vez por medio de MATLAB para generar la base de datos, la cual será consultada mediante la interfaz desarrollada en C++ la cual recibirá los valores de los índices de las mediciones realizadas por el robot durante su operación.

El entrenamiento del sistema o la evaluación de los parámetros de las superficies difusas se llevó a cabo durante la fase de experimentación, previamente se menciona el ajuste de parámetros para las funciones de membresía del sensor frontal, así como la evaluación de las reglas, ya que las reglas enunciadas aquí son el resultado de una serie de experimentos y de distintas reglas propuestas durante el desarrollo de este trabajo que fueron modificadas debido a que no brindaban el comportamiento deseado del robot.

4.9.-Red de Petri

En los apartados anteriores se plantearon algoritmos para que el robot sea capaz de localizar el objetivo en un entorno libre de obstáculos mediante la cámara y se propuso un sistema difuso para que el robot pueda realizar la exploración de un entorno con obstáculos usando la información obtenida de los sensores con los que cuenta. Pero ninguno de estos dos algoritmos es capaz de realizar el objetivo propuesto en este trabajo, por esta razón es necesario utilizar alguna otra herramienta que nos permita integrarlos para que de esta manera el robot pueda completar su tarea de manera satisfactoria. Para eso se emplea una red de Petri ya que nos permite describir el comportamiento deseado del sistema y administrar los recursos, como son el robot, la cámara, los movimientos del robot, etc. y también administrar los algoritmos, es decir elegir el momento en que interviene el sistema de inferencia difuso. La siguiente red es la que se utilizó en este trabajo:

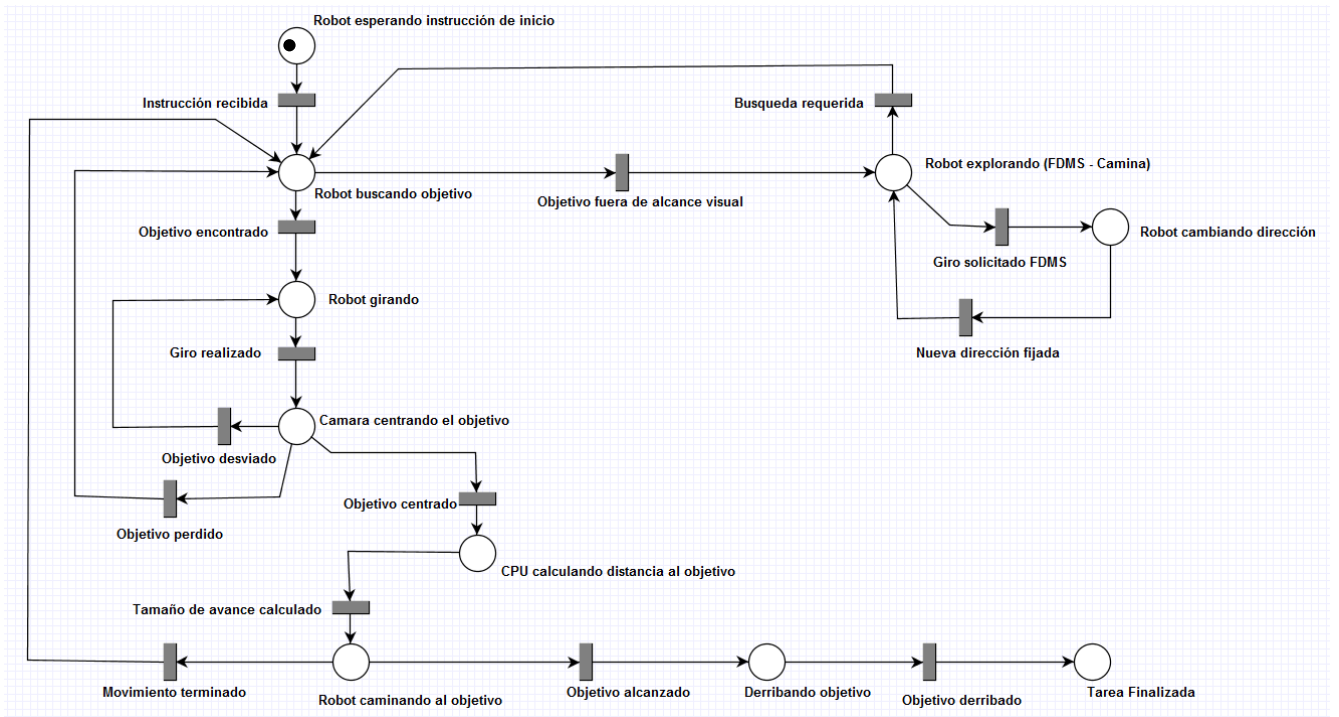


Figura 68.- Red de Petri usada en la navegación del robot humanoide.

La red de Petri está compuesta por 10 lugares y 14 transiciones, los lugares representan acciones del robot o recursos, las transiciones representan algunas condiciones alcanzadas por el robot o estados referentes a las acciones que realiza el robot. Se puede ver que la rama de la izquierda es prácticamente el mismo algoritmo que ya se describió para que el robot centre el objetivo y se desplace hacia él. Y que la rama de la derecha son las acciones que debe realizar cuando está explorando, es decir el sistema difuso de inferencia. Las transiciones que se encargan de relacionar estas dos ramas son: "Objetivo fuera de alcance visual" que significa que después del paneo de la cámara no se ha encontrado el objetivo por lo que es necesario que el robot comience a explorar el entorno para buscarlo, y la otra transición es "búsqueda requerida" mientras el robot está explorando solo adquiere información del entorno mediante los sensores ópticos, lo cual es suficiente para desplazarse pero de esta manera no puede localizar el objetivo por esta razón es que el robot debe realizar paneos del entorno en algunas ocasiones, se eligió que el robot realizará un paneo después de un caminado largo o un caminado medio y de esta manera logre localizar el objetivo, esta es la condición que dispara esta transición.

La rdP tiene un marcaje inicial en el cual el robot esta encendido y listo para comenzar su tarea en cuanto reciba la orden desde la computadora, esta orden debe ser dada desde la interface, usando una instrucción de voz o mediante un botón.

4.9.1.- Ecuación de la red de Petri

Como se menciona en el marco teórico es de gran interés conocer la ecuación que describe el comportamiento de las redes de Petri, porque nos permiten conocer sus marcajes futuros, a continuación se calcula la ecuación relacionada a la red de Petri usada en este trabajo. Primero expresemos la red en términos de lugares y transiciones, es decir, en vez de referirnos a los lugares y las transiciones por sus descripción como se muestran en figura 68, utilizaremos P seguido un número para los lugares y T seguido un número para las transiciones como se muestran enseguida:

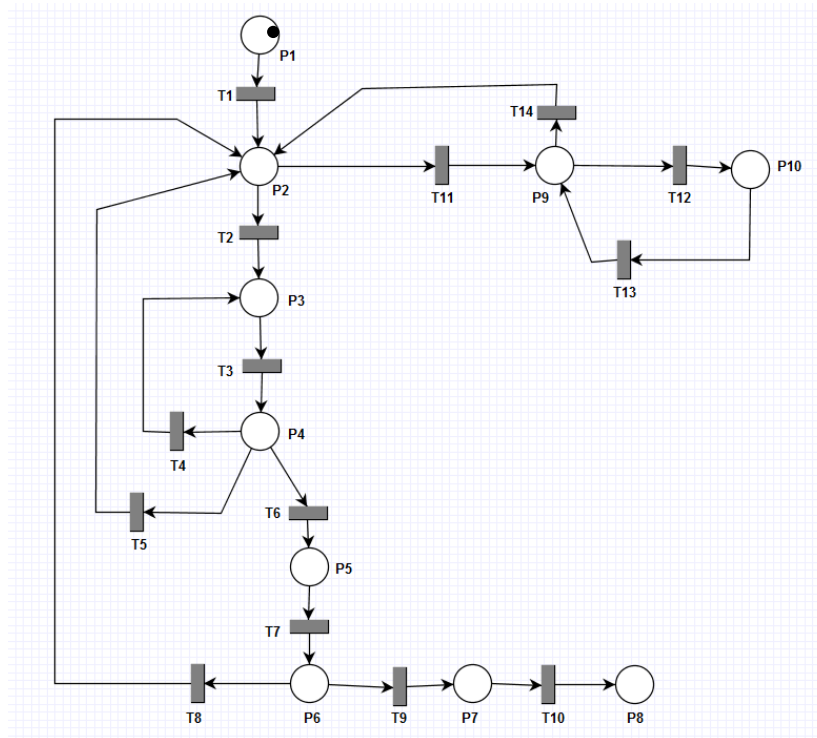


Figura 69.- Lugares y transiciones de la red de Petri.

Lo primero que se calculó fueron las matrices Pre y Pos, las cuales tienen dimensiones de 10 filas por 14 columnas.

$$\mathbf{W}^- = \begin{array}{c|cccccccccccc|c}
 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} & T_{13} & T_{14} & \\
 P_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_1 \\
 P_2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & P_2 \\
 P_3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_3 \\
 P_4 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_4 \\
 P_5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_5 \\
 P_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & P_6 \\
 P_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & P_7 \\
 P_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_8 \\
 P_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & P_9 \\
 P_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & P_{10}
 \end{array}$$

En la fila del lugar 8 todos sus elementos son ceros porque este lugar no es la entrada de ninguna transición, lo que significa que este lugar es un sumidero, lo que quiere decir que solo consume tokens o marcas y que es un punto muerto de la red, esto ocurre así porque este lugar es el que representa que el robot ha alcanzado el objetivo y que lo ha derribado lo que da por terminada su tarea.

$$\mathbf{W}^+ = \begin{array}{c|cccccccccccc|c}
 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} & T_{13} & T_{14} & \\
 P_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_1 \\
 P_2 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & P_2 \\
 P_3 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_3 \\
 P_4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_4 \\
 P_5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_5 \\
 P_6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_6 \\
 P_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & P_7 \\
 P_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & P_8 \\
 P_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & P_9 \\
 P_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & P_{10}
 \end{array}$$

$$\mathbf{W} = \mathbf{W}^+ - \mathbf{W}^-$$

Ya que se tienen los valores de Pre y Post entonces se calcula la matriz de incidencia al realizar la resta de estas dos matrices, Post menos Pre:

$$\mathbf{W} = \begin{array}{c|cccccccccccc|c}
 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} & T_{13} & T_{14} & \\
 \hline
 P_1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
 P_2 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 & \\
 P_3 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
 P_4 & 0 & 0 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
 P_5 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
 P_6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & \\
 P_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & \\
 P_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \\
 P_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & \\
 P_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 &
 \end{array}$$

Entonces como el marcaje inicial es:

$$\mathbf{m}_0 = \begin{array}{c|c}
 1 & P_1 \\
 0 & P_2 \\
 0 & P_3 \\
 0 & P_4 \\
 0 & P_5 \\
 0 & P_6 \\
 0 & P_7 \\
 0 & P_8 \\
 0 & P_9 \\
 0 & P_{10}
 \end{array}$$

Finalmente obtenemos la ecuación fundamental de la red, que está dada por la ecuación:

$$\mathbf{m}_k = \mathbf{m}_i + \mathbf{W} \cdot \mathbf{s}$$

Donde \mathbf{s} es el vector característico de dimensión igual al número de transiciones. Y representa las transiciones que se dispararán para llegar al siguiente marcaje.

Al calcular el marcaje \mathbf{m}_1 entonces el valor de \mathbf{s} es el siguiente:

$$\mathbf{s}^T = \begin{array}{c|cccccccccccc|c}
 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} & T_{13} & T_{14} & \\
 \hline
 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &
 \end{array}$$

Donde se obtiene:

$$(\mathbf{W} \cdot \mathbf{s})^T = \begin{array}{c|cccccccc|c}
 & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 & P_{10} & \\
 \hline
 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &
 \end{array}$$

Lo anterior nos dice que al activa la transición T_1 lo que ocurrirá en la red con relación al marcaje anterior será que el lugar uno perderá un token y que el lugar dos ganara un token, al calcular la suma para obtener a m_1 es igual a :

$$m_1 = \begin{vmatrix} P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 & P_{10} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

Lo que gráficamente es:

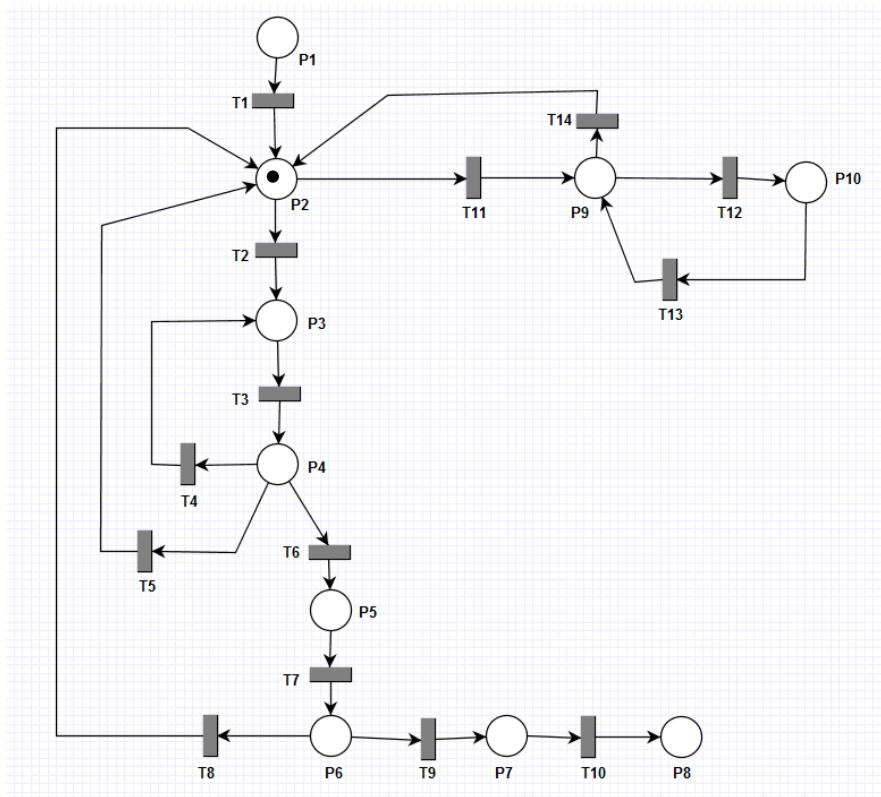


Figura 70.- Red de Petri con marcaje m_1 .

En este estado de la red, las transiciones 2 y 11 se encuentran habilitada y listas para dispararse, para calcular el marcaje que resultara de disparar una u otra utilizamos nuevamente la ecuación fundamental donde ahora tenemos los vectores característicos:

$$s_2^T = \begin{vmatrix} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} & T_{13} & T_{14} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

$$s_{11}^T = \begin{vmatrix} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} & T_{13} & T_{14} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{vmatrix}$$

Los marcajes finales que pueden alcanzarse son los siguientes, primero es necesario multiplicar los vectores característicos con la matriz de incidencia lo que da como resultado:

$$(\mathbf{W} \cdot \mathbf{s}_2)^T = \begin{vmatrix} P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 & P_{10} \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

$$(\mathbf{W} \cdot \mathbf{s}_{11})^T = \begin{vmatrix} P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 & P_{10} \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{vmatrix}$$

Al sumar lo anterior con el marcaje \mathbf{m}_1 se tiene:

$$\mathbf{m}_2 = \begin{vmatrix} P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 & P_{10} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

$$\mathbf{m}_{11} = \begin{vmatrix} P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 & P_{10} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{vmatrix}$$

De los ejemplos anteriores para calcular los marcajes de la red muestran algunas de las ventajas de utilizar las Redes de Petri, ya que para conocer el comportamiento futuro del sistema solo es necesario realizar algunas operaciones simples de matrices. Además los marcajes obtenidos dejan ver que la red tiene otras propiedades, ya que en los marcajes calculados solo hay un token en un lugar a la vez, a continuación se hace un análisis de estas propiedades y su relación con el sistema que describe la rdP.

4.9.2.- Propiedades de la red de Petri

Una de las propiedades que puede verse a simple vista de la red de Petri usada en este trabajo es que se trata de una red **pura** ya que no tiene bucles en sus lugares. La red es **ordinaria** debido a que todos los pesos de los arcos son uno, estas propiedades están estrechamente relacionadas a que la red también sea **segura** ya que todos los lugares están acotados a tener un solo token a la vez, esta propiedad es debido a que el sistema que se está describiendo es secuencial, debido a que el robot no es capaz de realizar tareas en paralelo, además está limitado a realizar las tareas una a la vez, comienza una tarea descrita por un lugar y hasta que la termina es posible disparar la transición para realizar la siguiente tarea.

La red es **cuasi-viva** ya que todas sus transiciones pueden ser disparadas por lo menos una vez durante la evolución de la red, esta propiedad describe que el robot siempre tiene la posibilidad de encontrar el objetivo y derribarlo ya que existe una evolución en la red que hace que la transición T_9 se habilite y pueda ser disparada, en otras palabras la posibilidad de que el robot finalice su tarea siempre se encuentra latente, no se conoce el tiempo en el que podría ocurrir pero siempre se mantiene como un estado al cual puede evolucionar la red. Las implicaciones de la propiedad anterior son grandes y demuestran el poder de las redes de Petri a pesar de su simplicidad, si bien lo anterior no nos asegura que el robot finalizara de manera correcta todos los experimentos, esto debido a que la red no modela algunos eventos que pueden afectar el desempeño del robot como son fallas en la comunicación o en el caminar, que se caiga o que se agote la batería, lo que si nos asegura es que si no se presentan estas fallas y que se encuentra un objeto como el que está buscando en el área de trabajo entonces el robot lo encontrara y finalizara su tarea en un tiempo finito, aunque este pueda ser muy grande. Lo anterior significa que el algoritmo siempre convergerá a solucionar la tarea, lo cual es algo deseado en todos los algoritmos y que para determinarse requiere en algunos casos de un análisis más profundo, pero que por medio de las redes de Petri basta con determinar si existe una secuencia de transiciones mediante la cual se alcanza el lugar deseado, lo cual puede ser calculado de manera sencilla.

La red es **cuasi-viva** porque si se dispara la transición T_{10} esto implica que la red se bloquee y llegue a un punto muerto y se dice que la red está bloqueada, ya no puede evolucionar mas allá de ese lugar, usualmente se desea evitar los puntos muertos en las redes, pero para la aplicación que esta describiendo es correcto ya que si el robot ha finalizado su tarea no tiene sentido que siga buscando o siga explorando el entorno. Por lo tanto que exista este punto muerto en la red es correcto y algo deseado.

La red de Petri puede ser reinicializada al marcaje m_1 desde cualquiera de los lugares siempre y cuando no se dispare la transición T_9 , lo que significa que siempre existe una secuencia de transiciones que llevara al robot a realizar una búsqueda del objeto. La cual es la tarea principal ya que de esta depende que el robot se está moviendo hacia el lugar correcto y corrija el camino en caso de que se haya encontrado el objetivo y se haya desviado el camino, o que se mantenga explorando el entorno.

Las propiedades anteriores son referentes al diseño de la red, las cuales son determinadas de acuerdo a la composición de los lugares y transiciones de la red, a continuación mediante el cálculo de los invariantes se encontraron de manera analítica otras propiedades y se demostraran algunas de las propiedades antes mencionadas.

Para los P-invariantes se debe satisfacer la ecuación:

$$x^T \cdot W = 0$$

Donde:

$$x^T = \begin{matrix} & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 & P_{10} \\ & X_1 & X_2 & X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9 & X_{10} \end{matrix} \quad \left| \right.$$

Y

$$W = \begin{matrix} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} & T_{13} & T_{14} \\ \begin{matrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \end{matrix} & \begin{matrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \\ P_8 \\ P_9 \\ P_{10} \end{matrix} \end{matrix}$$

Lo que da como resultado:

$$x^T \cdot W = \begin{matrix} -X_1 + X_2 \\ -X_2 + X_3 \\ -X_3 + X_4 \\ X_3 - X_4 \\ X_2 - X_4 \\ -X_4 + X_5 \\ -X_5 + X_6 \\ X_2 - X_6 \\ -X_6 + X_7 \\ -X_7 + X_8 \\ -X_2 + X_9 \\ -X_9 + X_{10} \\ X_9 - X_{10} \\ X_2 - X_9 \end{matrix} = \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

De donde se puede deducir que :

$$X_1 = X_2 = X_3 = X_4 = X_5 = X_6 = X_7 = X_8 = X_9 = X_{10} = \mathbf{a}$$

Donde \mathbf{a} es cualquier número entero positivo para cumplir con las condiciones de los P-invariantes, debido a que $\mathbf{a} = \mathbf{c} \cdot \mathbf{i}$ donde \mathbf{i} es el vector unitario formado solo por unos, como todos los valores de \mathbf{a} son composiciones de \mathbf{c} por \mathbf{i} , es decir que son linealmente dependientes y solo es de interés la solución única, por lo tanto el valor del invariante es \mathbf{i} :

$$\mathbf{x}^T = \begin{array}{c} P_1 \quad P_2 \quad P_3 \quad P_4 \quad P_5 \quad P_6 \quad P_7 \quad P_8 \quad P_9 \quad P_{10} \\ \left| \begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array} \right| \end{array}$$

Se puede ver que si multiplicamos el P-invariante \mathbf{x}^T por cualquier marcaje alcanzable por la red \mathbf{m}_i el resultado siempre será 1, lo que nos dice que solo habrá un token en toda la red para cualquier marcaje, lo que significa que el robot solo puede realizar una tarea a la vez de acuerdo a su comportamiento secuencial. Lo anterior se refiere a que la red es **conservativa**, ya que $\mathbf{x} > 0$ tal que la ecuación $\mathbf{x}^T \cdot \mathbf{W} = 0$. Lo cual nos dice que el número de tokens se mantiene constante en la red, en este caso siempre habrá un solo token.

El P-invariante encontrado es tal que cumple $\mathbf{x} > 0$ lo que además hace cierto $\mathbf{x}^T \cdot \mathbf{W} \leq 0$ por lo tanto rdP es **estructuralmente acotada**. De manera más precisa lo anterior demuestra que la red de Petri de este trabajo es **segura** ya que estructuralmente es 1-acotada.

Ahora para los T-invariantes se debe satisfacer la ecuación:

$$\mathbf{W} \cdot \mathbf{y} = 0$$

Donde:

$$\mathbf{y}^T = \begin{array}{c} T_1 \quad T_2 \quad T_3 \quad T_4 \quad T_5 \quad T_6 \quad T_7 \quad T_8 \quad T_9 \quad T_{10} \quad T_{11} \quad T_{12} \quad T_{13} \quad T_{14} \\ \left| \begin{array}{cccccccccccccc} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 & Y_6 & Y_7 & Y_8 & Y_9 & Y_{10} & Y_{11} & Y_{12} & Y_{13} & Y_{14} \end{array} \right| \end{array}$$

Y

$$\mathbf{W} = \begin{array}{c|cccccccccccc|c}
 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} & T_{13} & T_{14} & \\
 \hline
 P_1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
 P_2 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 & \\
 P_3 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
 P_4 & 0 & 0 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
 P_5 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
 P_6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & \\
 P_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & \\
 P_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \\
 P_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & \\
 P_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 &
 \end{array}$$

Lo cual da como resultado:

$$\mathbf{W} \cdot \mathbf{y} = \begin{array}{c|c}
 & \begin{array}{c}
 -Y_1 \\
 Y_1 + Y_2 + Y_5 + Y_8 - Y_{11} + Y_{14} \\
 Y_2 - Y_3 + Y_4 \\
 Y_3 - Y_4 - Y_5 - Y_6 \\
 Y_6 - Y_7 \\
 Y_7 - Y_8 - Y_9 \\
 Y_9 - Y_{10} \\
 Y_{10} \\
 Y_{11} - Y_{12} + Y_{13} - Y_{14} \\
 Y_{12} - Y_{13}
 \end{array} \\
 \hline
 & \begin{array}{c}
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0
 \end{array}
 \end{array} = \begin{array}{c}
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0
 \end{array}$$

Que al resolverse queda:

$$\mathbf{y}^T = \begin{array}{c|cccccccccccc|c}
 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} & T_{13} & T_{14} & \\
 \hline
 & 0 & 0 & 1 & 1 & -1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 &
 \end{array}$$

Lo cual no es un T-invariante. La única solución que cumple con $\mathbf{W} \cdot \mathbf{y} = 0$ es la solución trivial cuando $\mathbf{y} = 0$, lo que implica que no tiene T-invariantes lo que significa que la red de Petri no es **reinicializable**. Es decir que existe un punto muerto en la red, lo cual en esta ocasión es una situación deseada.

4.10.-Interfaz

Se desarrolló una interfaz en C++ la cual se encarga de procesar la información recibida del robot bioloid así como de acceder a la implementación del sistema difuso de inferencia y de indicarle al robot que acción debe realizar. En la implementación de la interfaz se utilizaron las librerías proporcionadas por el fabricante del robot bioloid para la comunicación Zigbee, esta librería está compuesta por cuatro métodos, uno para inicializar la comunicación, uno para cerrar la comunicación Zigbee con el robot, uno para enviar y otro para recibir información.

La información que recibe la interfaz del robot es la siguiente:

- La medición del sensor frontal.
- La medición del sensor izquierdo.
- La medición del sensor derecho.
- El número de pixeles de la región de color segmentada por la cámara.
- El centro de la región de color en el eje X.
- El centro de la región de color en el eje Y.
- El estatus en que se encuentra el robot, como puede ser que ya encontró el objetivo, que está explorando el entorno, que ha finalizado la tarea o que ha perdido de vista el objetivo.

La información recibida, las mediciones de los sensores y el número de pixeles, es procesada mediante sus ecuaciones de linealización para poder desplegar los valores en centímetros en la pantalla para que los pueda ver el usuario. Otra información que se muestra en la pantalla es una aproximación de lo que capta la cámara, ya que usando los valores de la región en "X" y "Y" se puede dibujar una región con un número de pixeles aproximado al medido por la cámara.

A continuación se muestra la pantalla de la interfaz:



Figura 71.- Interfaz desarrollada en C++ para controlar al robot bioloid.

Lo primero que hace la interfaz es establecer la comunicación con el robot, si la comunicación se estableció correctamente un mensaje en la esquina superior derecha lo indicara, lo siguiente es darle la instrucción de inicio al robot, equivalente a la transición T_1 de la red de Petri para que comience a funcionar el robot, esto se hace al presionar el botón de inicio, , la primera tarea es realizar un paneo al área de trabajo, una vez hecho esto comienza el intercambio de información entre el robot y la computadora. La interfaz una vez que ha recibido la información del robot, dependiendo del estado en que se encuentre el robot va a realizar las siguientes acciones:

- En caso de que el objetivo haya sido localizado y este centrado con relación al robot, calcula la distancia a la cual se encuentra el objetivo e indicarle el caminado que debe realizar el robot.
- En cambio si el robot se encuentra explorando el entorno utilizara los valores de distancia entregados por los sensores como índices para consultar la base de conocimiento formada por el sistema difuso de inferencia y de esta manera tomar una decisión sobre cuál será la instrucción que se le enviara al robot para continuar con la exploración.

“Navegación de un robot humanoide mediante redes de Petri y lógica difusa”

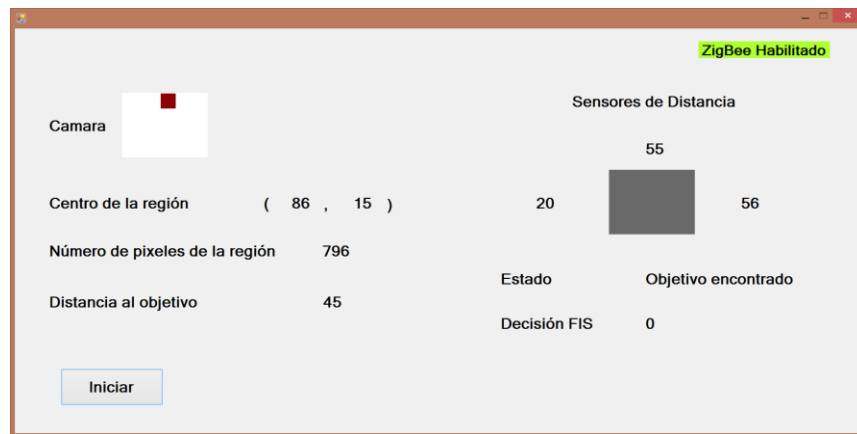


Figura 72.- Ejemplo 1 de la interfaz en operación cuando el objetivo ha sido encontrado.

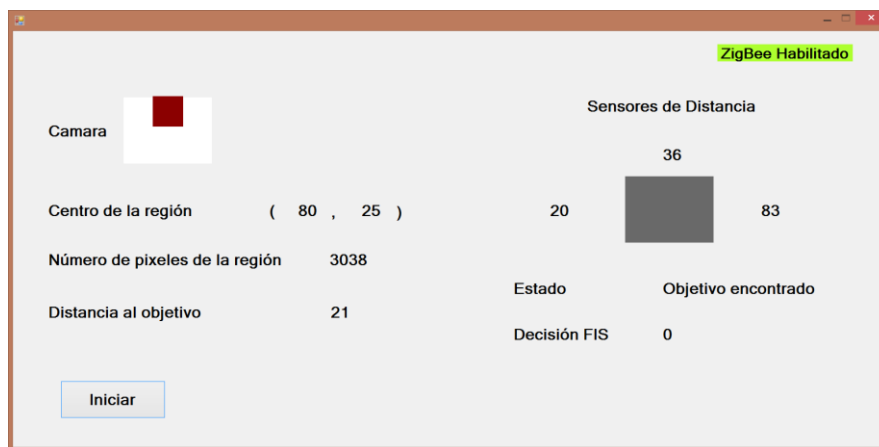
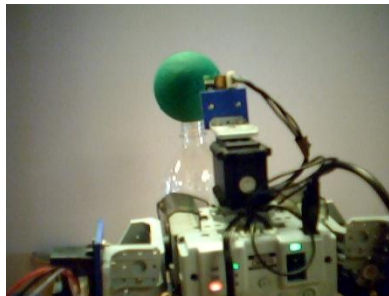


Figura 73.- Ejemplo 2 de la interfaz en operación cuando el objeto ha sido encontrado.

“Navegación de un robot humanoide mediante redes de Petri y lógica difusa”

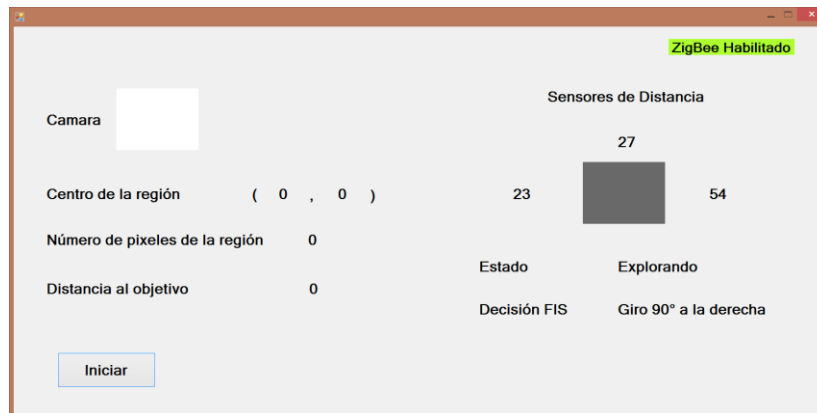
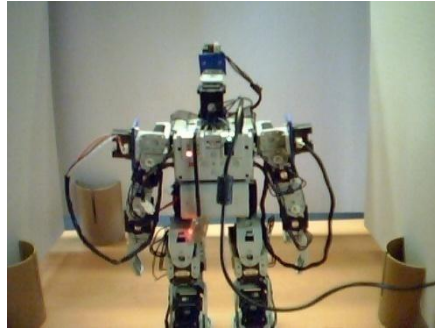
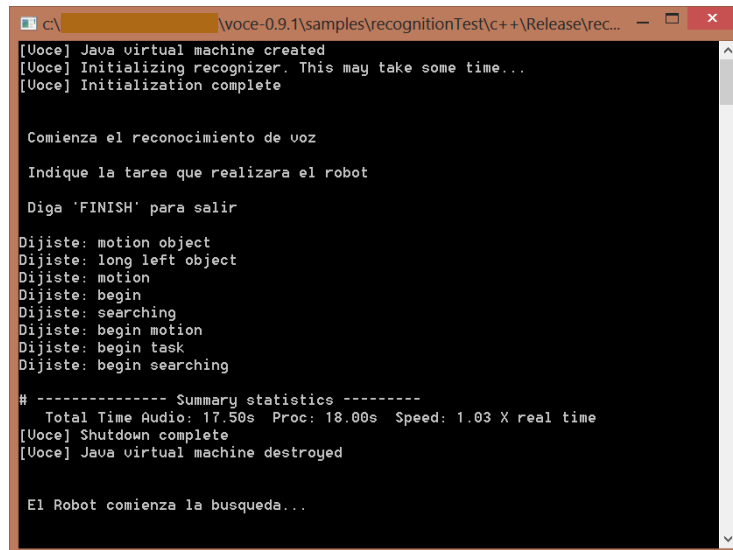


Figura 74.- Ejemplo de la interfaz en operación cuando el robot está explorando el entorno.

Otra posibilidad de comenzar con la operación del robot es mediante un comando de voz, que es prácticamente lo mismo que presionar el botón de inicio, ya que previo a que inicie la operación de la interfaz se lanza una aplicación desarrollada en C++ llamada VOCE, que es una API para el reconocimiento de voz desarrollada para funcionar en JAVA o en C++ que utiliza internamente CMU sphinx4 y FreeTTS. Por medio de estos módulos internos esta API es capaz de hacer un reconocimiento voz de un gran número de palabras en inglés.

Básicamente el funcionamiento de esta API es mediante una biblioteca la cual contiene métodos para inicializar el corpus el cual utiliza un archivo que contiene las palabras que pueden ser reconocidas, y después dentro de la aplicación de C++ se llama al método que hace el reconocimiento de voz y entrega como resultado la palabra más parecida, dentro del archivo antes mencionado, a la palabra dicha al micrófono, usualmente hace un reconocimiento correcto si la palabra se dijo de manera adecuada y se utiliza un micrófono. Una de las ventajas de este sistema es que no requiere de un entrenamiento solo es necesario consultar un archivo que contiene las palabras para las que ya ha sido entrenado y usar estas, todas las palabras están en inglés y deben de ser dichas en inglés. Algunas de las palabras que se seleccionaron para que pudieran ser reconocidas son "task", "motion", "begin", "searching", "start", "left", "right", "long", "short", "medium", "find" y "object", esta API incluso puede reconocer oraciones o secuencias de palabras. Una vez que esta aplicación reconoció la orden correcta en este caso "Begin Searching" se cierra e inicia la interfaz de operación del robot que ya fue descrita.



```
c:\voce-0.9.1\samples\recognitionTest\c++\Release\rec...
[Voce] Java virtual machine created
[Voce] Initializing recognizer. This may take some time...
[Voce] Initialization complete

Comienza el reconocimiento de voz

Indique la tarea que realizara el robot

Diga 'FINISH' para salir

Dijiste: motion object
Dijiste: long left object
Dijiste: motion
Dijiste: begin
Dijiste: searching
Dijiste: begin motion
Dijiste: begin task
Dijiste: begin searching

# ----- Summary statistics -----
Total Time Audio: 17.50s Proc: 18.00s Speed: 1.03 X real time
[Voce] Shutdown complete
[Voce] Java virtual machine destroyed

El Robot comienza la busqueda...
```

Figura 75.- Aplicación VOCE para el reconocimiento de ordenes por voz.

Capítulo 5

Resultados experimentales y análisis

En este capítulo se muestran las pruebas realizadas, se explican las características del entorno controlado que se habilitó y se describen las configuraciones utilizadas para probar el desempeño del modelo híbrido. Se mencionan cuales fueron los problemas presentes al realizar las pruebas y se muestran imágenes del funcionamiento del robot al buscar el objetivo en las distintas configuraciones.

5.1.-Área de trabajo

Para evaluar el desempeño del sistema difuso de inferencia y la red de Petri propuestos en este trabajo se habilitó un ambiente controlado donde se realizaron las pruebas necesarias. Se utilizaron 4 configuraciones distintas del entorno, donde el robot debía de desplazarse sin chocar contra las paredes u obstáculos y encontrar el objetivo buscado. A continuación se explicaran las características del área de trabajo y del objetivo que busca el robot. Después se muestran los resultados de los experimentos realizados en cada una de las distintas configuraciones.

Para delimitar el área de trabajo y como obstáculos se utilizaron placas de cartón, las cuales no interfieren con el proceso de reconocimiento de color de la cámara ya que son de un color distinto al del objetivo buscado por el robot. Las placas de cartón tienen dimensiones de 76 cm de largo por 51 cm de alto, pero se cortaron algunas para obtener segmentos más cortos, que fueron usados como paredes internas, es decir obstáculos en algunas de las configuraciones.



Figura 76.- Materiales con los que se habilitó el área de trabajo.

El objetivo que buscara el robot en todas las pruebas realizadas es una esfera de unicel de color verde de 8 cm de diámetro, la cual es colocada en un pedestal para que sea más fácil que el robot pueda localizarla.



Figura 77.- Objetivo del robot.

Las dimensiones del área de trabajo para las configuraciones 1 y 2 son 1.2 m x 1.3 m, una superficie total de 1.5 m². La configuración 1 solo está delimitada en su contorno por las laminas de cartón, lo que se considera por lo tanto un ambiente libre de obstáculos. Mientras que la configuración 2 agrega un par de paredes al entorno con la intención de ocultar el objetivo del robot para que realice una exploración del área de trabajo.



Figura 78.- Área de trabajo de las configuraciones 1 y 2.

5.2.-Configuración 1

Las pruebas en esta configuración consistieron en colocar al robot en algún lugar del área de trabajo y en algún punto del entorno se coloca la esfera. En estos experimentos la posición hacia donde este mirando la cámara no es importante, ya que se realiza un paneo de casi 360°. En esta configuración el robot no tiene ningún problema para finalizar su tarea, salvo cuando las condiciones de luz no son las adecuadas, pero en todos los casos donde las condiciones controladas se mantienen el robot logra llegar a donde está la esfera y derribarla. En esta configuración se realizaron 15 experimentos los cuales fueron satisfactorios. A continuación se muestran imágenes donde se puede apreciar el funcionamiento del robot en esta configuración.

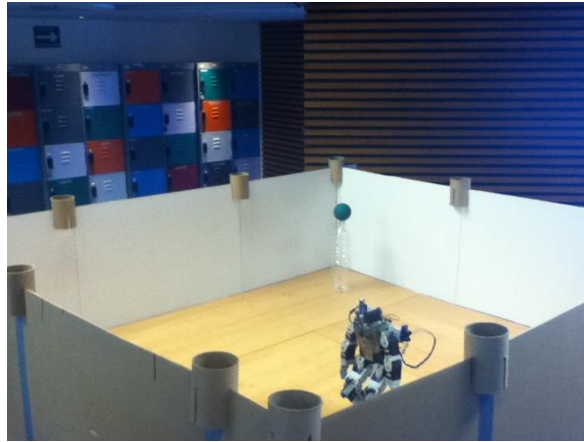


Figura 79.- Área de trabajo configuración 1.

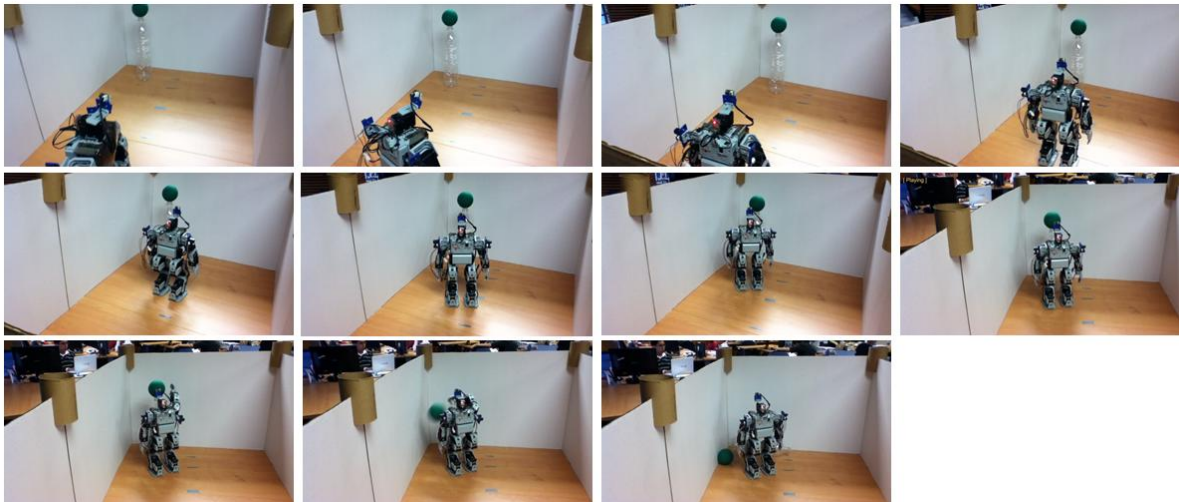


Figura 80.- Ejemplo del funcionamiento del robot en la configuración 1.

En la siguiente secuencia se muestra el funcionamiento del robot cuando comienza su funcionamiento con la cámara mirando hacia las paredes. El tiempo que le lleva al robot finalizar su tarea en esta configuración es de entre 2 a 3 minutos.

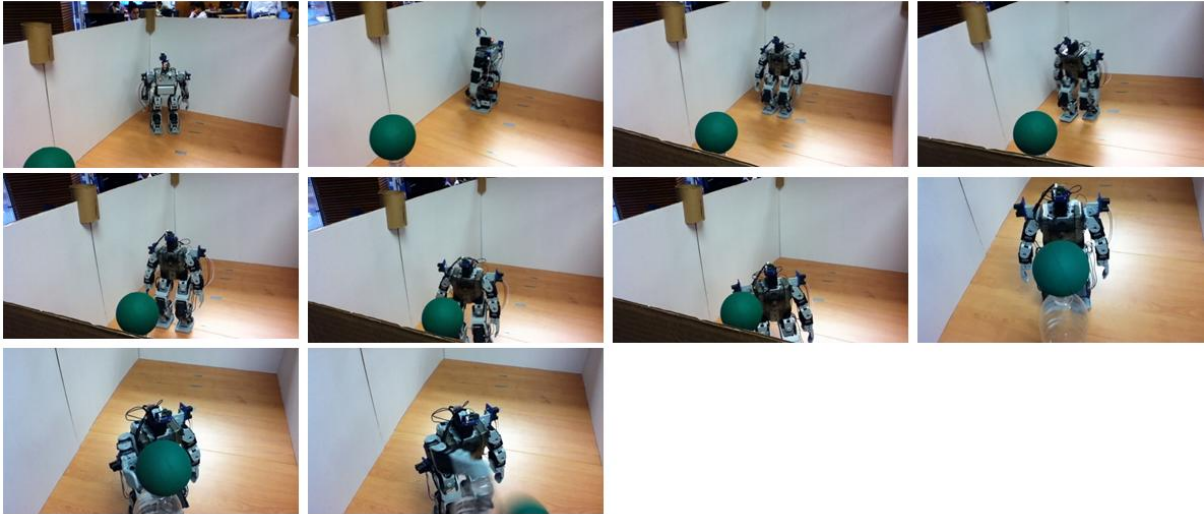


Figura 81.- Otro ejemplo del funcionamiento del robot en la configuración 1.

5.3.- Configuración 2

En esta configuración el robot es colocado en el punto A que puede verse en la figura 82 y el objetivo se coloca en el punto B, por lo tanto el robot no es capaz de ver al objetivo desde el principio así que necesita comenzar a explorar el entorno ya que no cuenta con ninguna información previa ni del espacio de trabajo ni de la localización del objetivo.

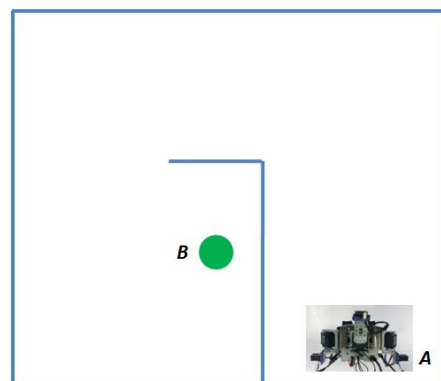


Figura 82.- Diagrama configuración 2.

En estas pruebas fue necesario considerar ciertas características con las que debe cumplir el área de trabajo para que el robot se desplace de manera correcta, estas son que los pasillos por los cuales caminará el robot tengan un ancho de por lo menos 50 o 60 cm, debido al rango de los sensores el funcionamiento del robot será mejor en espacios amplios, además debido a que el robot desvía su dirección al caminar se genera un error el cual no puede ser medido, ni estimado, ni tampoco predecible ya que es generado por el caminar del robot y la superficie en la cual se desplaza por lo tanto hay ocasiones en las cuales se presenta y hay otras en las que no. Este error no es tan grande como para hacer que el robot cambie su dirección más de 90°, por lo que el mismo sistema difuso es capaz de irlo compensando durante la operación del robot, lo cual ocasiona que el robot haga el recorrido en un mayor tiempo y que a veces no logre finalizar la tarea debido a que ha desviado demasiado su trayectoria.

Los experimentos se llevaron a cabo colocando al robot en el punto A con la cámara mirando hacia la pared más lejana, los resultados obtenidos se muestran a continuación.



Figura 83.-Ejemplo 1 del funcionamiento del robot en la configuración 2.

Este recorrido le lleva al robot entre 4 a 5 minutos. En este caso se realizaron solo 12 pruebas con 10 éxitos y 2 fallas debidas a desvíos durante el desplazamiento del robot.

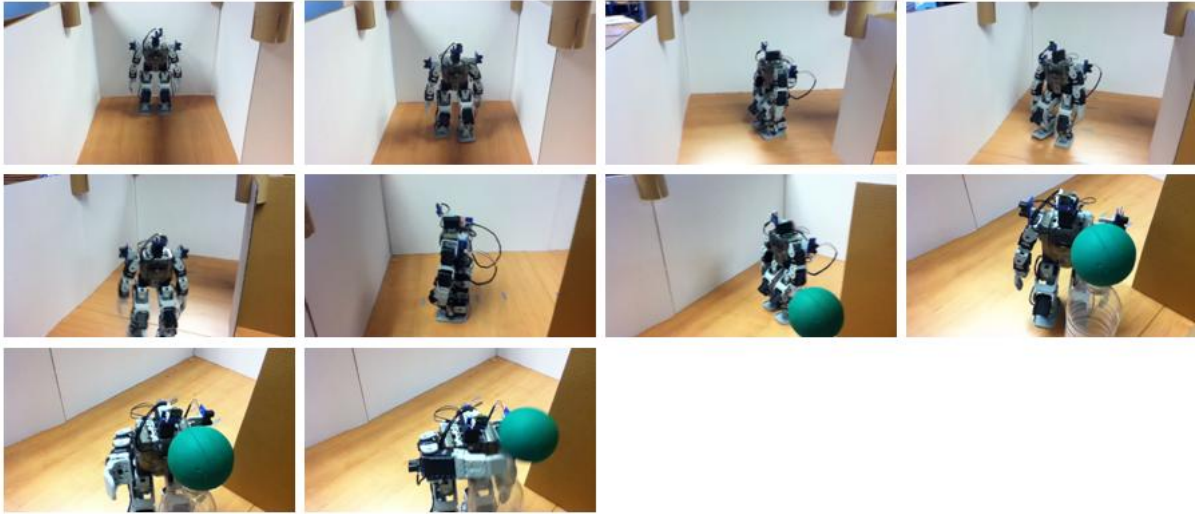


Figura 84.- Ejemplo 2 del funcionamiento del robot en la configuración 2.

Para las configuraciones 3 y 4 el área de trabajo es mayor, ya que sus dimensiones son de 2 m x 1.9 m, lo que da una superficie de 3.8 m², esto se hizo así para poner a prueba al sistema difuso y a la red de Petri a un entorno en el cual el robot tuviera que realizar más giros y desplazarse distancias más grandes.



Figura 85.- Área de trabajo para las configuraciones 3 y 4.

5.4.- Configuración 3

En esta configuración se pone a prueba el sistema difuso del robot, ya que para llegar al objetivo el robot debe explorar el entorno, el cual requiere que realice giro a la izquierda y a la derecha, a diferencia de la configuración 2, donde solo se realizan giros a la izquierda. En estos experimentos se presentaron otras fallas ajenas al trabajo propuesto en este documento, ya que la comunicación se interrumpía en algunas ocasiones impidiéndole al robot finalizar la tarea. El error generado por el caminado prevalecieron y siguió causando fallas. Una cuestión más que debió ser considerada fue la presencia de ruido, es decir regiones de color espurias durante la exploración que interferían con el funcionamiento del robot, por ello tuvo que definirse un número mínimo de píxeles para que pudiera considerarse que se había encontrado el objetivo, es decir que el robot solo puede localizar el objetivo si ha alcanzado una distancia igual o menor a la equivalente al número de píxeles mínimo de acuerdo a la ecuación para calcular la distancia con la cámara. El número mínimo de píxeles es 150.

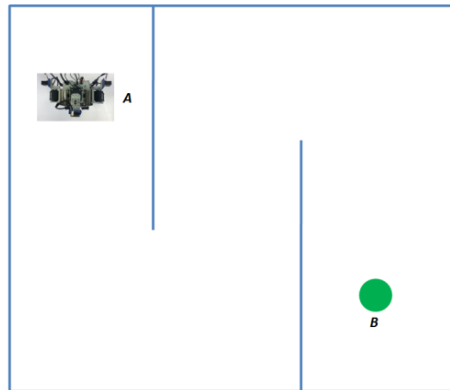


Figura 86.- Diagrama configuración 3.

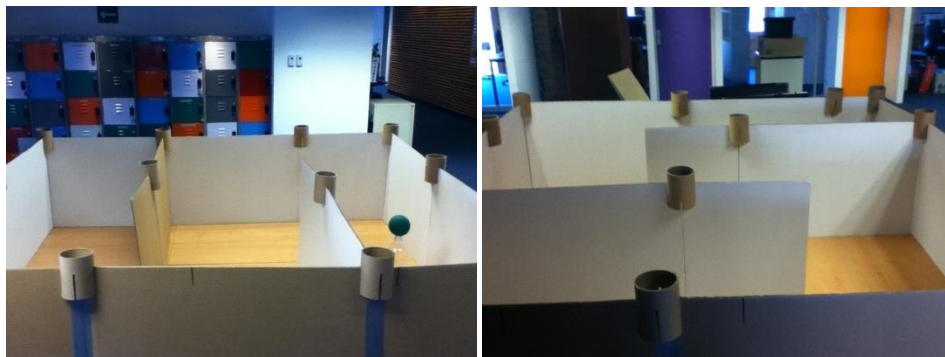


Figura 87.- Configuración 3.

Otra condición que afecto el desempeño del robot fue que, el robot no tiene conciencia de las dimensiones que ocupa en el espacio, debido al limitado número de sensores con los que cuenta, por lo cual el robot en algunas ocasiones percibe que existe espacio libre al frente pero en realidad es un espacio angosto por el cual no puede atravesar sin que ocurra una colisión.

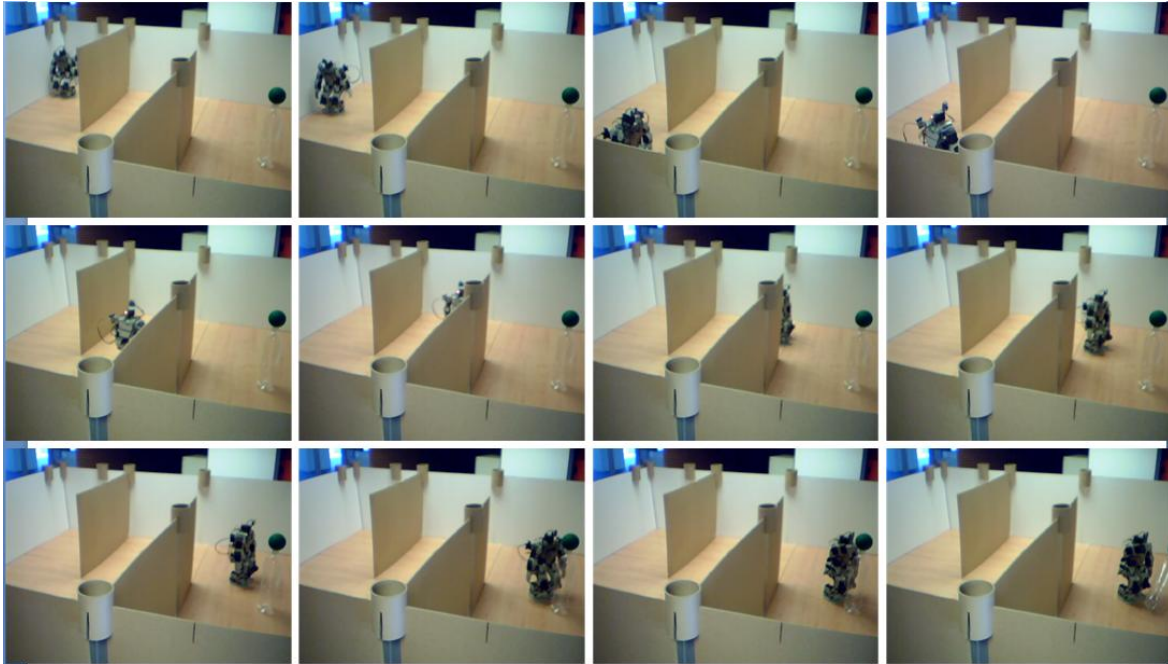


Figura 88.- Ejemplo 1 del funcionamiento del robot en la configuración 3.

Estos recorridos le llevan al robot entre 9 a 10 minutos, esto debido a que es necesario que realice paneos cuando las condiciones así lo indican lo cual le lleva alrededor de 30 segundos, durante este tiempo el robot debe permanecer quieto ya que no puede caminar al mismo tiempo, además el caminado del robot es lento.

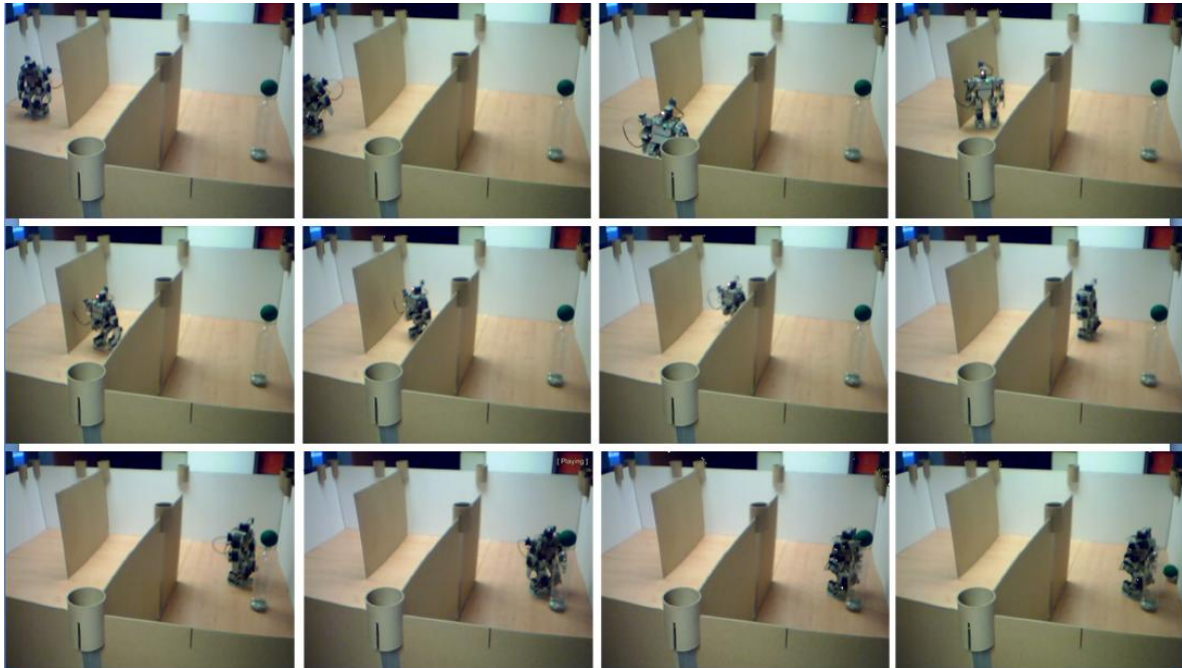


Figura 89.- Ejemplo 2 del funcionamiento del robot en la configuración 3.

En esta configuración se realizaron 15 experimentos en los cuales solo se obtuvieron 10 recorridos completos, donde 3 fallas se debieron al caminar deficiente del robot y dos a que el robot alcanzo una posición donde el sistema difuso de inferencia comenzó a oscilar, es decir que le indicaba que girara a la derecha y luego a la izquierda, lo cual es un comportamiento no deseado pero que es ocasionado por las reglas que se definieron.

5.5.- Configuración 4

La configuración 4 es similar a la usada en (8) que puede verse en la figura 5, se decidió utilizar esta configuración para comparar el desempeño de la solución propuesta en este trabajo contra la realizada en (8). Aunque una de las principales diferencias es el uso de robots móviles con ruedas en (8) y que en esta tesis se utilizo un robot humanoide, lo que hace difícil realizar una comparación objetiva del desempeño de los robots en ambos trabajos. Obviamente el tiempo no puede ser un factor a comparar ya que el recorrido del robot humanoide es de 9 a 10 minutos mientras que el del robot móvil con ruedas es menor.

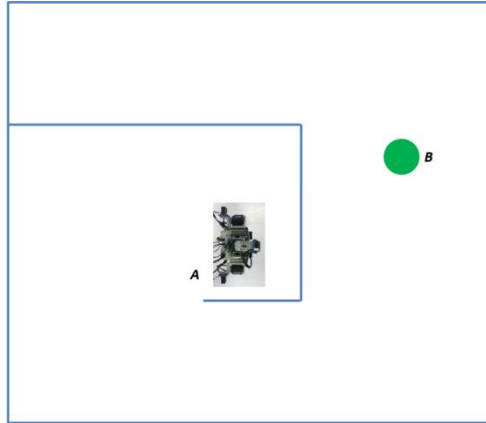


Figura 90.- Diagrama configuración 4.

Sin embargo se puede ver que el comportamiento es similar y que el desplazamiento se realiza en una trayectoria muy similar. No es posible determinar con precisión que tan similares son debido a factores de escala, es decir la proporción que existe entre el tamaño del robot y el área de trabajo, lo que repercute en la distancia total recorrida por el robot para alcanzar su objetivo.

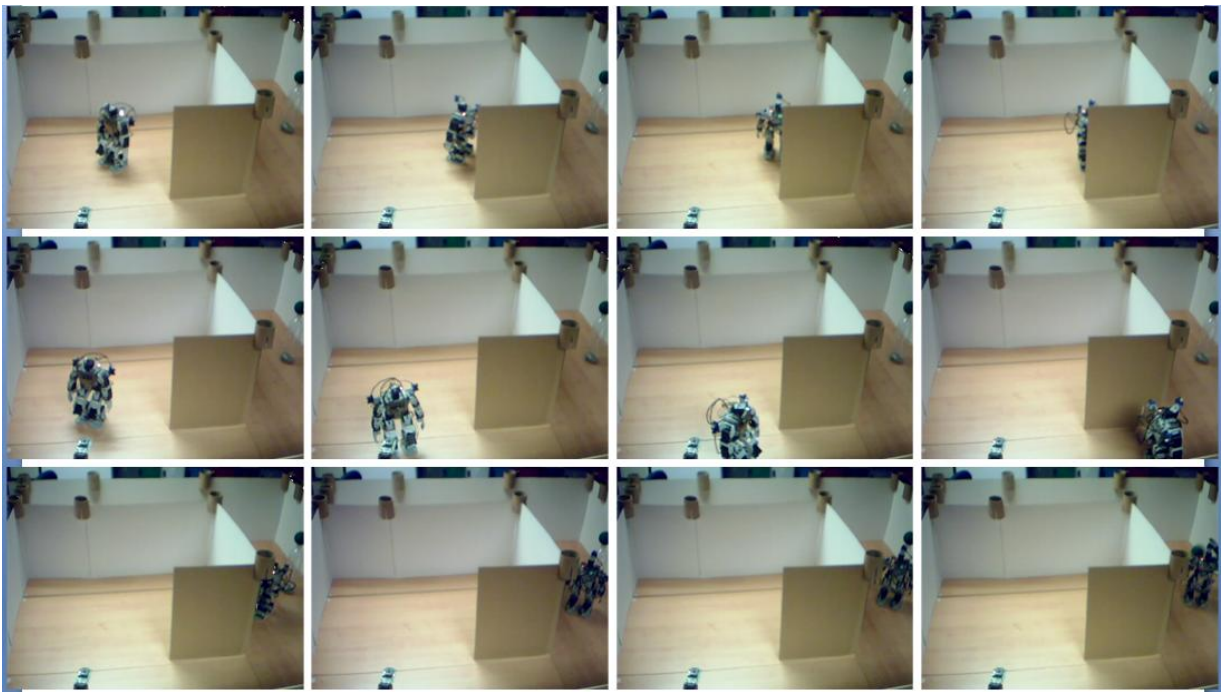


Figura 91.- Ejemplo 1 del funcionamiento del robot en la configuración 4.

De este experimento se realizaron 10 experimentos de los cuales solo 6 fueron recorridos exitosos. Durante estos experimentos se presentaron problemas con la comunicación, se interrumpía o era intermitente. Sin embargo no se presentaron problemas con el funcionamiento del sistema difuso.

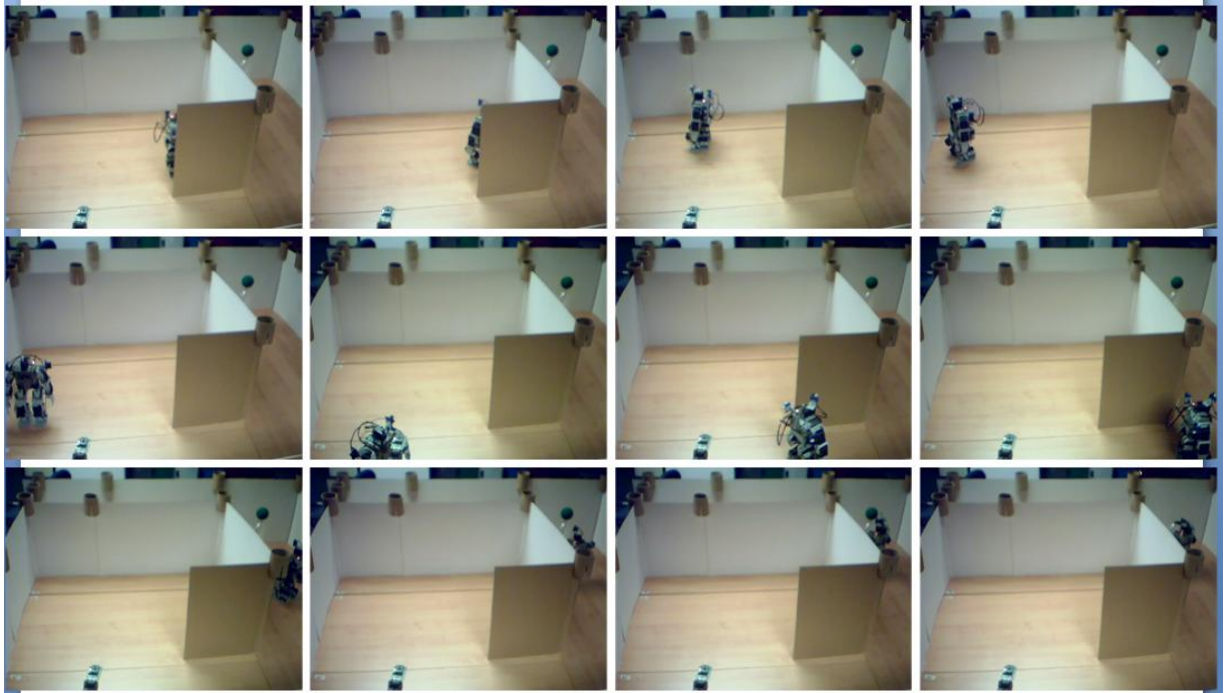


Figura 92.- Ejemplo 2 del funcionamiento del robot en la configuración 4.

Capítulo 6

Conclusiones, trabajos futuros y recomendaciones.

En este capítulo se concluye con las experiencias, logros y aportaciones que resultaron de este trabajo, así como los posibles trabajos a futuro que pueden ser llevados a cabo para complementar lo que se desarrolló en esta tesis, al final se dan algunas recomendaciones para quienes estén interesados en realizar un trabajo similar o continuar con una investigación que enriquezca lo aquí propuesto.

6.1.- Conclusiones

Se logró cumplir con el objetivo propuesto en esta tesis, ya que se formulo un método de navegación basado en lógica difusa y redes de Petri. Para alcanzar esta meta se tomaron las ventajas que brindan las herramientas usadas para compensar algunos problemas. El sistema de inferencia difusa permitió generar una alternativa de movimiento para el robot sin la necesidad de calcular el modelo cinemático o encontrar las ecuaciones diferenciales que rigen su comportamiento, en cambio solo fue necesario describir en forma de reglas el comportamiento deseado del robot.

Las redes de Petri nos permitieron describir de manera clara y simple el comportamiento deseado del sistema, en este caso es la responsable de administrar recursos y algoritmos. Otra alternativa de solución pudo haber sido tener más entradas en el sistema difuso para hacerlo robusto, lo que lo vuelve más complejo al mismo tiempo, y que de esta manera fuese capaz de realizar la tarea de localizar el objetivo solo, pero en vez de eso se propuso un FIS menos complejo y se utilizo a la red de Petri para combinarlo con otras acciones y tareas que le permitió llevar a cabo la tarea de manera satisfactoria.

El modelo híbrido propuesto en este trabajo resultó ser flexible y adaptable a la plataforma en donde fue probado, ya que el sistema difuso es capaz de tolerar el error generado durante el caminado del robot y continuar con la tarea de manera correcta, mientras que la red de Petri le da la posibilidad de siempre poder finalizar la tarea.

Además la solución no está limitado solo a la plataforma donde fue probado, ya que puede ser adaptada o pueden agregársele algunos otros componentes que puedan enriquecerla, como puede ser utilizar otros sensores en vez de la cámara HAVIMO para determinar el camino que deba seguir el robot, emplear un arreglo de mas sensores que midan distancia para obtener información del entorno, etc., ya que lo que se tendría que modificar sería la red de Petri, es decir agregar transiciones o lugares de acuerdo al comportamiento deseado, o bien modificar las entradas o las reglas del FIS.

Si bien el número de recorridos exitosos realizados por el robot no fue tan bueno como se esperaba, esto no quiere decir que la solución propuesta en este trabajo para el problema de navegación sea mala. El funcionamiento del sistema difuso y la red de Petri es correcto lo que genera los malos resultados en gran medida es la plataforma en donde se probó, ya que genera condiciones que son ajenas al modelo híbrido propuesto, que repercuten en los resultados. Aunque estadísticamente parezca que el funcionamiento del modelo híbrido no es bueno, si se descartan los errores ocasionados por la plataforma y solo se analiza el desempeño del modelo híbrido se obtiene una eficiencia mayor al 80% de las veces. El desempeño del modelo fue probado en diferentes ambientes controlados, lo que le da gran versatilidad, algo deseado en los robots móviles.

Finalmente algo que no se puede pasar por alto es que no hay antecedentes de un trabajo similar en el centro de investigación en computación, lo cual es valioso al proponer una solución novedosa a un problema que es de gran interés para la comunidad de investigadores y lograr obtener un modelo cuyos resultados experimentales fueron buenos, lo cual puede ser la base para generar algunos trabajos futuros que puedan culminar en algún momento con una aportación sustancial en el campo de los robots móviles.

6.2.- Trabajo futuro

Uno de los principales trabajos a futuro es probar el modelo híbrido en otras plataformas, es decir en un robot móvil con ruedas, en un robot humanoide de última generación o incluso en un quadrotor. Ya que debido a su simplicidad, lo que lo vuelve fácil de comprender, es sencillo realizar algunos ajustes para poder adaptarlo a una nueva plataforma, sin pérdida de generalidad, ya que el diseño no fue hecho en función de los materiales y el equipo con que se contaba, sino enfocado a resolver el problema de la navegación en móviles.

Otro trabajo a futuro es la reconstrucción en la computadora del entorno en el cual se desplaza el robot, es decir generar un mapa del espacio donde se está desplazando el robot con la finalidad de que mediante el uso de un sistema de coordenadas el robot pueda regresar a su posición inicial o alcanzar algún otro punto del espacio que ya ha recorrido, es decir que pueda realizar una exploración para describir el entorno desconocido. Esta recolección de información del ambiente en el cual se desplaza resulta útil en la exploración de ambientes de difícil acceso o que representen un riesgo para las personas, como podría ser el caso en que el robot encontrara una ruta de evacuación en un siniestro para que las personas afectadas la sigan.

Otro trabajo a futuro puede ser agregarle al modelo alguna fase de aprendizaje, es decir que se ajusten los parámetros de las funciones de membresía de acuerdo al entorno en el cual se desplaza el robot para mejorar su desempeño. O el aprendizaje de rutas por las cuales ya ha transitado, e incluso la generación o adaptación de las reglas difusas para que el robot pueda tener un mejor funcionamiento y no solo se mueva como un seguidor de paredes.

6.3.- Recomendaciones

Una recomendación para los trabajos futuros es conocer bien los elementos con los que se trabajara, cuáles son sus ventajas y desventajas, para de esta forma fijar los alcances adecuados y proponer una solución acorde a lo que se puede alcanzar. También es importante entender que un robot no es una máquina inteligente, sino que es un conjunto de sensores y actuadores que interactuaran de acuerdo a como se le indique o se programe, por lo tanto la inteligencia o las tareas que pueda realizar dependen del modelo utilizado o la manera en que este sea aplicado.

El uso de las redes de Petri es recomendable, aunque no sea parte de la solución, para comprender mejor el sistema y la manera en que interactúan los recursos o las tareas que estén involucradas en el sistema, ya que al conocer el flujo o el comportamiento del sistema es posible prever fallas y proponer soluciones o tratar de evitarlas.

Al utilizar los sistemas difusos se debe tratar de mantener la menor complejidad posible en tanto al número de entradas del cual depende el número de reglas, lo cual puede hacer que obtener las salidas del sistema sea muy lento.

Referencias

1. **Katic, Dusko and Vukobratovic, Miomir.** *Survey of Intelligent Control Techniques for Humanoid Robots.* Journal of Intelligent and Robotic Systems, 37: 117-141, 2003.
2. **Willis, Chris.** www.androidworld.com, 2013. [Online]
3. **Honda Motor Co., Ltd.** <http://www.world.honda.com/ASIMO/>, 2013. [Online]
4. **Marchese, S., Muscato, G. and Virk, G. S.** *Dynamically stable trajectory synthesis for a biped robot during the single-support phase.* Proc. of the 2001 IEEE/ASME Internat. Conf. on Advanced Intelligent Mechatronics, 1:953 - 958, 2001.
5. **Zimmermann, H.-J.** *Fuzzy Sets Theory and Its Applications.* Kluwer Academic Publishers, 1990.
6. **Terano, T., Asai, K. and Sugeno, M.** *Fuzzy Systems Theory and Its Applications.* Boston, USA : Academic, 1992.
7. **Chandra Mohanta, Jagadish, Ramakrushna Parhi, Dayal and Kumar Patel, Saroj.** *Path planning strategy for autonomous mobile robot navigation using Petri-GA optimisation.* Computers and Electrical Engineering, 37: 1058 - 1070, 2011.
8. **Chandra Mohanta, Jagadish and Ramakrushna Parhia, Dayal.** *Navigational control of several mobile robotic agents using Petri-potential-fuzzy hybrid controller.* Applied Soft Computing, 11: 3546 - 3557, 2011.
9. **Murata, Tadao.** *Petri nets: Properties, Analysis and Applications.* Proceedings of the IEEE, 77:514 - 580, 1989.
10. **Milutinovic, Dejan and Lima, Pedro.** *Petri net models of robotic tasks.* Proceedings of the 2002 IEEE International Conference on Robotics & Automation, 2:4059 - 4064, 2002.
11. **Perera, L. P., Carvalho, L. P. and Guedes Soares, C.** *Fuzzy logic based decision making system for collision avoidance of ocean navigation under critical collision condition.* Journal of Marine Science and Technology, 16:84 - 99, 2010.
12. **Ziparo, V. A., et al.** *Petri net plans - A framework for collaboration and coordination in multi-robot systems.* Auton Agent Multi-Agent Syst, 23:344 - 383, 2010.
13. **Calisi, D., et al.** *Multi-objective exploration and search for autonomous rescue robots.* Journal of Field Robotics, 24:763 - 777, 2007.
14. **Farinelli, A., et al.** *Assignment of dynamically perceived tasks by token passing in multi-robot systems.* Proceedings of the IEEE - Special issue on multi-robot systems, 94:1271 -1288.
15. **David, René and Alla, Hassane.** *Discrete, Continuous, and Hybrid Petri Nets.* s.l. : Springer, 2010. 978-3-642-10668-2.

16. **Lee H., Kwang.** *First course on fuzzy theory and applications.* s.l. : Springer, 2005. 3-540-22988-4.
17. **ROBOTIS.** e-Manual. <http://support.robotis.com/en/>, 2010. [Online]
18. **Bid, Hamid.** *HaViMo2 Image Processing Module.*
<http://robosavvy.com/RoboSavvyPages/Support/Hamid/HaViMo2.pdf>, 2010. [Online]
19. **Wai, Rong-Jong, Liu, Chia-Ming and Lin, You-Wei.** *Robust path tracking control of mobile robot via dynamic petri recurrent fuzzy neural network.* *Soft Comput*, 15:743 - 767, 2010.
20. **ROBOTIS CO., LTD.** *Zigbee Module.*
http://support.robotis.com/en/product/auxdevice/communication/zigbee_manual.htm, 2006 [Online]
21. **Roger Jang, Jyh-Shing, Sun, Chuen-Tsai and Mizutani, Eiji.** *Neuro-fuzzy and soft computing.* s.l. : Prentice hall, 1997. 0-13-261066-3.
22. **Sivanandam, S. N., Sumathi, S. and Deepa, S. N.** *Introduction to Fuzzy Logic using MATLAB.* s.l. : Springer, 2007. 10 3-540-35780-7.

Anexos

Biblioteca para comunicación ZigBee.

```
#ifndef _ZIGBEE_HEADER
#define _ZIGBEE_HEADER

#ifdef __cplusplus
extern "C" {
#endif

////////// device control methods //////////
int __stdcall zgb_initialize( int devIndex );
void __stdcall zgb_terminate();

////////// communication methods //////////
int __stdcall zgb_tx_data(int data);
int __stdcall zgb_rx_check();
int __stdcall zgb_rx_data();

////////// define RC-100 button key value //////////
#define RC100_BTN_U      (1)
#define RC100_BTN_D      (2)
#define RC100_BTN_L      (4)
#define RC100_BTN_R      (8)
#define RC100_BTN_1      (16)
#define RC100_BTN_2      (32)
#define RC100_BTN_3      (64)
#define RC100_BTN_4      (128)
#define RC100_BTN_5      (256)
#define RC100_BTN_6      (512)

#ifdef __cplusplus
}
#endif

#endif
```

Código de la interfaz

```
#pragma once
#pragma comment(lib, "zigbee.lib")

namespace Proyecto {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Summary for Form1
    ///
    /// WARNING: If you change the name of this class, you will need to
change the
    /// 'Resource File Name' property for the managed resource
compiler tool
    /// associated with all .resx files this class depends on.
Otherwise,
    /// the designers will not be able to interact properly
with localized
    /// resources associated with this form.
    /// </summary>
    public ref class Form1 : public System::Windows::Forms::Form
    {
    public:
        Form1(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
        }

    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~Form1()
        {
            if (components)
            {
                delete components;
            }
        }

    private: System::Windows::Forms::Label^ lb11;
    protected:

    protected:
```

```
private:
    /// <summary>
    /// Required designer variable.
    /// Defulat setting
    #define DEFAULT_PORTNUM      4 // COM
    #define TIMEOUT_TIME_SENSOR  100 // msec
    #define TIMEOUT_TIME_CAM     200 // msec

    int RX;
    int ss;

    int TxData, RxData;
    int s;
    int cam;
    int ox,oy; //Valores de la posicion inicial de la imagen de
la camara
    int act; //variable que decide cuando debe realizar una
accion el robot
    int status;
    int fk,rk,lk; //Sensores
    int fk1,rk1,lk1; //Sensores en k menos uno
    int fk2,rk2,lk2; //Sensores en k menos dos
    int d_f,d_r,d_l; //Diferencia entre la medicion actual y la
anterior de los sensores
    int i_f,i_r,i_l; //INDICES DE LOS SENSORES
    int x,y,p; //Datos de la camara
    int t;

private: System::Windows::Forms::Label^ left;
private: System::Windows::Forms::Label^ front;
private: System::Windows::Forms::Label^ right;
private: System::Windows::Forms::PictureBox^ pictureBox1;

private: System::Windows::Forms::PictureBox^ pictureBox2;
private: System::Windows::Forms::Label^ Cy;

private: System::Windows::Forms::Label^ Cx;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::Label^ Tp;

private: System::Windows::Forms::Label^ label1;
private: System::Windows::Forms::Label^ label2;
private: System::Windows::Forms::Button^ button3;
private: System::Windows::Forms::Button^ button4;
private: System::Windows::Forms::Button^ button5;
private: System::Windows::Forms::Button^ button6;

private: System::Windows::Forms::Label^ label4;

private: System::Windows::Forms::Label^ label5;
private: System::Windows::Forms::Timer^ timer4;
```



```
private: System::Windows::Forms::Button^ button7;
private: System::Windows::Forms::Label^ Status;
private: System::Windows::Forms::Label^ dist;
private: System::Windows::Forms::Label^ label6;
private: System::Windows::Forms::Label^ label7;
private: System::Windows::Forms::TreeView^ treeView1;
private: System::Windows::Forms::Button^ button1;
private: System::Windows::Forms::Button^ button2;
private: System::Windows::Forms::Label^ label8;
private: System::Windows::Forms::Label^ label9;
private: System::Windows::Forms::Label^ label10;
private: System::Windows::Forms::Label^ label11;
private: System::Windows::Forms::Label^ label12;
private: System::Windows::Forms::Label^ label13;
private: System::ComponentModel::IContainer^ components;
    /// </summary>

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        this->components = (gcnew
System::ComponentModel::Container());
        this->lbl1 = (gcnew System::Windows::Forms::Label());
        this->left = (gcnew System::Windows::Forms::Label());
        this->front = (gcnew System::Windows::Forms::Label());
        this->right = (gcnew System::Windows::Forms::Label());
        this->pictureBox1 = (gcnew
System::Windows::Forms::PictureBox());
        this->pictureBox2 = (gcnew
System::Windows::Forms::PictureBox());
        this->Cy = (gcnew System::Windows::Forms::Label());
        this->Cx = (gcnew System::Windows::Forms::Label());
        this->label3 = (gcnew System::Windows::Forms::Label());
        this->Tp = (gcnew System::Windows::Forms::Label());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->label2 = (gcnew System::Windows::Forms::Label());
        this->button3 = (gcnew
System::Windows::Forms::Button());
        this->button4 = (gcnew
System::Windows::Forms::Button());
        this->button5 = (gcnew
System::Windows::Forms::Button());
        this->button6 = (gcnew
System::Windows::Forms::Button());
        this->label4 = (gcnew System::Windows::Forms::Label());
        this->label5 = (gcnew System::Windows::Forms::Label());
        this->timer4 = (gcnew
System::Windows::Forms::Timer(this->components));
        this->button7 = (gcnew
System::Windows::Forms::Button());
        this->Status = (gcnew System::Windows::Forms::Label());
        this->dist = (gcnew System::Windows::Forms::Label());
```

```
        this->label6 = (gcnew System::Windows::Forms::Label());
        this->label7 = (gcnew System::Windows::Forms::Label());
        this->treeView1 = (gcnew
System::Windows::Forms::TreeView());
        this->button1 = (gcnew
System::Windows::Forms::Button());
        this->button2 = (gcnew
System::Windows::Forms::Button());
        this->label8 = (gcnew System::Windows::Forms::Label());
        this->label9 = (gcnew System::Windows::Forms::Label());
        this->label10 = (gcnew
System::Windows::Forms::Label());
        this->label11 = (gcnew
System::Windows::Forms::Label());
        this->label12 = (gcnew
System::Windows::Forms::Label());
        this->label13 = (gcnew
System::Windows::Forms::Label());

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox1))->BeginInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox2))->BeginInit();
        this->SuspendLayout();
        //
        // lbl1
        //
        this->lbl1->AutoSize = true;
        this->lbl1->BackColor =
System::Drawing::Color::FromArgb(static_cast<System::Int32>(static_cast<S
ystem::Byte>(255)),
static_cast<System::Int32>(static_cast<System::Byte>(128)),

        static_cast<System::Int32>(static_cast<System::Byte>(128)));
        this->lbl1->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 15.75F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->lbl1->Location = System::Drawing::Point(1275,
24);
        this->lbl1->Margin = System::Windows::Forms::Padding(4,
0, 4, 0);
        this->lbl1->Name = L"lbl1";
        this->lbl1->Size = System::Drawing::Size(292, 31);
        this->lbl1->TabIndex = 1;
        this->lbl1->Text = L"ZigBee Deshabilitado";
        //
        // left
        //
        this->left->AutoSize = true;
        this->left->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->left->Location = System::Drawing::Point(966,
309);
```

```
0, 4, 0);
    this->left->Margin = System::Windows::Forms::Padding(4,
    this->left->Name = L"left";
    this->left->Size = System::Drawing::Size(32, 32);
    this->left->TabIndex = 2;
    this->left->Text = L"0";
    //
    // front
    //
    this->front->AutoSize = true;
    this->front->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
    static_cast<System::Byte>(0)));
    this->front->Location = System::Drawing::Point(1170,
208);
    this->front->Margin =
System::Windows::Forms::Padding(4, 0, 4, 0);
    this->front->Name = L"front";
    this->front->Size = System::Drawing::Size(32, 32);
    this->front->TabIndex = 3;
    this->front->Text = L"0";
    //
    // right
    //
    this->right->AutoSize = true;
    this->right->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
    static_cast<System::Byte>(0)));
    this->right->Location = System::Drawing::Point(1348,
309);
    this->right->Margin =
System::Windows::Forms::Padding(4, 0, 4, 0);
    this->right->Name = L"right";
    this->right->Size = System::Drawing::Size(32, 32);
    this->right->TabIndex = 4;
    this->right->Text = L"0";
    //
    // pictureBox1
    //
    this->pictureBox1->BackColor =
System::Drawing::Color::DimGray;
    this->pictureBox1->Location =
System::Drawing::Point(1108, 263);
    this->pictureBox1->Margin =
System::Windows::Forms::Padding(4);
    this->pictureBox1->Name = L"pictureBox1";
    this->pictureBox1->Size = System::Drawing::Size(160,
120);
    this->pictureBox1->TabIndex = 5;
    this->pictureBox1->TabStop = false;
    //
    // pictureBox2
    //
    this->pictureBox2->BackColor =
System::Drawing::Color::White;
```

```
        this->pictureBox2->Location =
System::Drawing::Point(200, 120);
        this->pictureBox2->Margin =
System::Windows::Forms::Padding(4);
        this->pictureBox2->Name = L"pictureBox2";
        this->pictureBox2->Size = System::Drawing::Size(160,
120);

        this->pictureBox2->TabIndex = 7;
        this->pictureBox2->TabStop = false;
        //
        // Cy
        //
        this->Cy->AutoSize = true;
        this->Cy->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->Cy->Location = System::Drawing::Point(625, 309);
        this->Cy->Margin = System::Windows::Forms::Padding(4,
0, 4, 0);

        this->Cy->Name = L"Cy";
        this->Cy->Size = System::Drawing::Size(32, 32);
        this->Cy->TabIndex = 8;
        this->Cy->Text = L"0";
        //
        // Cx
        //
        this->Cx->AutoSize = true;
        this->Cx->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->Cx->Location = System::Drawing::Point(510, 309);
        this->Cx->Margin = System::Windows::Forms::Padding(4,
0, 4, 0);

        this->Cx->Name = L"Cx";
        this->Cx->Size = System::Drawing::Size(32, 32);
        this->Cx->TabIndex = 9;
        this->Cx->Text = L"0";
        //
        // label3
        //
        this->label3->AutoSize = true;
        this->label3->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->label3->Location = System::Drawing::Point(58,
309);

        this->label3->Margin =
System::Windows::Forms::Padding(4, 0, 4, 0);
        this->label3->Name = L"label3";
        this->label3->Size = System::Drawing::Size(275, 32);
        this->label3->TabIndex = 10;
        this->label3->Text = L"Centro de la región";
        //
        // Tp
```

```
//
this->Tp->AutoSize = true;
this->Tp->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
    static_cast<System::Byte>(0)));
this->Tp->Location = System::Drawing::Point(567, 398);
this->Tp->Margin = System::Windows::Forms::Padding(4,
0, 4, 0);

this->Tp->Name = L"Tp";
this->Tp->Size = System::Drawing::Size(32, 32);
this->Tp->TabIndex = 12;
this->Tp->Text = L"0";
//
// label1
//
this->label1->AutoSize = true;
this->label1->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
    static_cast<System::Byte>(0)));
this->label1->Location = System::Drawing::Point(58,
165);

this->label1->Margin =
System::Windows::Forms::Padding(4, 0, 4, 0);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(121, 32);
this->label1->TabIndex = 13;
this->label1->Text = L"Camara";
//
// label2
//
this->label2->AutoSize = true;
this->label2->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
    static_cast<System::Byte>(0)));
this->label2->Location = System::Drawing::Point(1033,
120);

this->label2->Margin =
System::Windows::Forms::Padding(4, 0, 4, 0);
this->label2->Name = L"label2";
this->label2->Size = System::Drawing::Size(320, 32);
this->label2->TabIndex = 14;
this->label2->Text = L"Sensores de Distancia";
this->label2->Click += gcnew System::EventHandler(this,
&Form1::label2_Click);
//
// button3
//
this->button3->Location = System::Drawing::Point(586,
673);

this->button3->Margin =
System::Windows::Forms::Padding(4);
this->button3->Name = L"button3";
this->button3->Size = System::Drawing::Size(100, 28);
this->button3->TabIndex = 15;
```

```
        this->button3->Text = L"LONG_F";
        this->button3->UseVisualStyleBackColor = true;
        this->button3->Visible = false;
        this->button3->Click += gcnew
System::EventHandler(this, &Form1::button3_Click);
        //
        // button4
        //
        this->button4->Location = System::Drawing::Point(1002,
673);
        this->button4->Margin =
System::Windows::Forms::Padding(4);
        this->button4->Name = L"button4";
        this->button4->Size = System::Drawing::Size(93, 28);
        this->button4->TabIndex = 16;
        this->button4->Text = L"TURN_LEFT";
        this->button4->UseVisualStyleBackColor = true;
        this->button4->Visible = false;
        this->button4->Click += gcnew
System::EventHandler(this, &Form1::button4_Click);
        //
        // button5
        //
        this->button5->Location = System::Drawing::Point(1103,
673);
        this->button5->Margin =
System::Windows::Forms::Padding(4);
        this->button5->Name = L"button5";
        this->button5->Size = System::Drawing::Size(84, 28);
        this->button5->TabIndex = 17;
        this->button5->Text = L"TURN_RIGHT";
        this->button5->UseVisualStyleBackColor = true;
        this->button5->Visible = false;
        this->button5->Click += gcnew
System::EventHandler(this, &Form1::button5_Click);
        //
        // button6
        //
        this->button6->Location = System::Drawing::Point(694,
673);
        this->button6->Margin =
System::Windows::Forms::Padding(4);
        this->button6->Name = L"button6";
        this->button6->Size = System::Drawing::Size(92, 28);
        this->button6->TabIndex = 18;
        this->button6->Text = L"MEDIUM_F";
        this->button6->UseVisualStyleBackColor = true;
        this->button6->Visible = false;
        this->button6->Click += gcnew
System::EventHandler(this, &Form1::button6_Click);
        //
        // label4
        //
        this->label4->BackColor =
System::Drawing::Color::DarkRed;
        this->label4->Enabled = false;
```

```

        this->label4->Location = System::Drawing::Point(276,
178);
        this->label4->Margin =
System::Windows::Forms::Padding(0);
        this->label4->Name = L"label4";
        this->label4->Size = System::Drawing::Size(13, 12);
        this->label4->TabIndex = 19;
        this->label4->Visible = false;
        this->label4->Click += gcnew System::EventHandler(this,
&Form1::label4_Click);
        //
        // label5
        //
        this->label5->AutoSize = true;
        this->label5->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->label5->Location = System::Drawing::Point(1170,
533);
        this->label5->Name = L"label5";
        this->label5->Size = System::Drawing::Size(32, 32);
        this->label5->TabIndex = 21;
        this->label5->Text = L"0";
        //
        // timer4
        //
        this->timer4->Interval = 1;
        this->timer4->Tick += gcnew System::EventHandler(this,
&Form1::timer4_Tick);
        //
        // button7
        //
        this->button7->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->button7->Location = System::Drawing::Point(86,
633);
        this->button7->Name = L"button7";
        this->button7->Size = System::Drawing::Size(190, 68);
        this->button7->TabIndex = 23;
        this->button7->Text = L"Iniciar";
        this->button7->UseVisualStyleBackColor = true;
        this->button7->Click += gcnew
System::EventHandler(this, &Form1::button7_Click_2);
        //
        // Status
        //
        this->Status->AutoSize = true;
        this->Status->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->Status->Location = System::Drawing::Point(1170,
450);
        this->Status->Name = L"Status";

```

```

        this->Status->Size = System::Drawing::Size(32, 32);
        this->Status->TabIndex = 24;
        this->Status->Text = L"0";
        //
        // dist
        //
        this->dist->AutoSize = true;
        this->dist->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->dist->Location = System::Drawing::Point(569,
493);

        this->dist->Name = L"dist";
        this->dist->Size = System::Drawing::Size(32, 32);
        this->dist->TabIndex = 25;
        this->dist->Text = L"0";
        //
        // label6
        //
        this->label6->AutoSize = true;
        this->label6->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->label6->Location = System::Drawing::Point(58,
398);

        this->label6->Name = L"label6";
        this->label6->Size = System::Drawing::Size(437, 32);
        this->label6->TabIndex = 26;
        this->label6->Text = L"Número de pixeles de la región";
        //
        // label7
        //
        this->label7->AutoSize = true;
        this->label7->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->label7->Location = System::Drawing::Point(1281,
533);

        this->label7->Name = L"label7";
        this->label7->Size = System::Drawing::Size(99, 32);
        this->label7->TabIndex = 27;
        this->label7->Text = L"label7";
        this->label7->Visible = false;
        //
        // treeView1
        //
        this->treeView1->Location = System::Drawing::Point(0,
949);

        this->treeView1->Name = L"treeView1";
        this->treeView1->Size = System::Drawing::Size(121, 97);
        this->treeView1->TabIndex = 28;
        //
        // button1
        //

```



```
        this->button1->Location = System::Drawing::Point(902,
673);
        this->button1->Margin =
System::Windows::Forms::Padding(4);
        this->button1->Name = L"button1";
        this->button1->Size = System::Drawing::Size(92, 28);
        this->button1->TabIndex = 30;
        this->button1->Text = L"TURN_180";
        this->button1->UseVisualStyleBackColor = true;
        this->button1->Visible = false;
        this->button1->Click += gcnew
System::EventHandler(this, &Form1::button1_Click_2);
        //
        // button2
        //
        this->button2->Location = System::Drawing::Point(794,
673);
        this->button2->Margin =
System::Windows::Forms::Padding(4);
        this->button2->Name = L"button2";
        this->button2->Size = System::Drawing::Size(100, 28);
        this->button2->TabIndex = 29;
        this->button2->Text = L"SHORT_F";
        this->button2->UseVisualStyleBackColor = true;
        this->button2->Visible = false;
        this->button2->Click += gcnew
System::EventHandler(this, &Form1::button2_Click_1);
        //
        // label8
        //
        this->label8->AutoSize = true;
        this->label8->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->label8->Location = System::Drawing::Point(458,
309);
        this->label8->Name = L"label8";
        this->label8->Size = System::Drawing::Size(25, 32);
        this->label8->TabIndex = 31;
        this->label8->Text = L"";
        //
        // label9
        //
        this->label9->AutoSize = true;
        this->label9->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->label9->Location = System::Drawing::Point(688,
311);
        this->label9->Name = L"label9";
        this->label9->Size = System::Drawing::Size(25, 32);
        this->label9->TabIndex = 32;
        this->label9->Text = L"";
        //
        // label10
```

```

        //
        this->label10->AutoSize = true;
        this->label10->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->label10->Location = System::Drawing::Point(569,
311);

        this->label10->Name = L"label10";
        this->label10->Size = System::Drawing::Size(24, 32);
        this->label10->TabIndex = 33;
        this->label10->Text = L",";
        //
        // label11
        //
        this->label11->AutoSize = true;
        this->label11->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->label11->Location = System::Drawing::Point(58,
493);

        this->label11->Name = L"label11";
        this->label11->Size = System::Drawing::Size(291, 32);
        this->label11->TabIndex = 34;
        this->label11->Text = L"Distancia al objetivo";
        //
        // label12
        //
        this->label12->AutoSize = true;
        this->label12->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->label12->Location = System::Drawing::Point(900,
450);

        this->label12->Margin =
System::Windows::Forms::Padding(4, 0, 4, 0);
        this->label12->Name = L"label12";
        this->label12->Size = System::Drawing::Size(110, 32);
        this->label12->TabIndex = 35;
        this->label12->Text = L"Estado";
        this->label12->Click += gcnew
System::EventHandler(this, &Form1::label12_Click);
        //
        // label13
        //
        this->label13->AutoSize = true;
        this->label13->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 16.2F,
System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(0)));
        this->label13->Location = System::Drawing::Point(900,
533);

        this->label13->Margin =
System::Windows::Forms::Padding(4, 0, 4, 0);
        this->label13->Name = L"label13";

```

```
        this->label13->Size = System::Drawing::Size(187, 32);
        this->label13->TabIndex = 36;
        this->label13->Text = L"Decisión FIS";
        //
        // Form1
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(8,
16);
        this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
        this->ClientSize = System::Drawing::Size(1582, 753);
        this->Controls->Add(this->label13);
        this->Controls->Add(this->label12);
        this->Controls->Add(this->label11);
        this->Controls->Add(this->label10);
        this->Controls->Add(this->label9);
        this->Controls->Add(this->label8);
        this->Controls->Add(this->button1);
        this->Controls->Add(this->button2);
        this->Controls->Add(this->treeView1);
        this->Controls->Add(this->label7);
        this->Controls->Add(this->label6);
        this->Controls->Add(this->dist);
        this->Controls->Add(this->Status);
        this->Controls->Add(this->button7);
        this->Controls->Add(this->label5);
        this->Controls->Add(this->label4);
        this->Controls->Add(this->button6);
        this->Controls->Add(this->button5);
        this->Controls->Add(this->button4);
        this->Controls->Add(this->button3);
        this->Controls->Add(this->label2);
        this->Controls->Add(this->label1);
        this->Controls->Add(this->Tp);
        this->Controls->Add(this->label3);
        this->Controls->Add(this->Cx);
        this->Controls->Add(this->Cy);
        this->Controls->Add(this->pictureBox2);
        this->Controls->Add(this->pictureBox1);
        this->Controls->Add(this->right);
        this->Controls->Add(this->front);
        this->Controls->Add(this->left);
        this->Controls->Add(this->lbl1);
        this->Margin = System::Windows::Forms::Padding(4);
        this->MaximizeBox = false;
        this->Name = L"Form1";
        this->Text = L" ";
        this->Load += gcnew System::EventHandler(this,
&Form1::Form1_Load);
        this->FormClosing += gcnew
System::Windows::Forms::FormClosingEventHandler(this,
&Form1::Form1_FormClosing);

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox1))->EndInit();
```

```
(cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox2))->EndInit();
    this->ResumeLayout(false);
    this->PerformLayout();
}
#pragma endregion
private: void cam_obj(int xx, int yy, int ip){
    ip=System::Int32(System::Math::Round(sqrt(double(ip))));
    label4->Size::set(System::Drawing::Size(ip,ip));
    xx=ox+xx-ip/2;
    yy=oy+yy-ip/2;
    label4->Location::set(Point(xx,yy));
}

//LOAD FORM
private: System::Void Form1_Load(System::Object^ sender,
System::EventArgs^ e) {

    // Open device
    //Form1::Size::set(System::Drawing::Size(1600,800));

    pictureBox2->Location::set(System::Drawing::Point(200,120));
    pictureBox2->Size::set(System::Drawing::Size(160,120));

    //lbl1->Location::set(System::Drawing::Point(1350,720));

    if( zgb_initialize(DEFAULT_PORTNUM) == 0 )
    {
        MessageBox::Show("ERROR en la comunicación!!!");
    }
    else{
        lbl1->Text="ZigBee Habilitado";
        lbl1->BackColor=System::Drawing::Color::GreenYellow;

    }
    ox=200;
    oy=120;
    y=20;
    fk=0;lk=0;rk=0;
    act=0;
}

//btn1 CLICK
private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e) {

}

private: System::Void Form1_FormClosing(System::Object^ sender,
System::Windows::Forms::FormClosingEventArgs^ e) {
    // Close device
    zgb_terminate();
}
```

```
private: System::Void button3_Click(System::Object^ sender,
System::EventArgs^ e) {
    int Tx=11;
    // Transmit data
    if(zgb_tx_data(Tx) == 0)
        MessageBox::Show( "Fallo la transmición!! " );
}

private: System::Void button6_Click(System::Object^ sender,
System::EventArgs^ e) {
    int Tx=12;
    // Transmit data
    if(zgb_tx_data(Tx) == 0)
        MessageBox::Show( "Fallo la transmición!! " );
}

private: System::Void button2_Click_1(System::Object^ sender,
System::EventArgs^ e) {
    int Tx=13;
    // Transmit data
    if(zgb_tx_data(Tx) == 0)
        MessageBox::Show( "Fallo la transmición!! " );
}

private: System::Void button4_Click(System::Object^ sender,
System::EventArgs^ e) {
    int Tx=14;
    // Transmit data
    if(zgb_tx_data(Tx) == 0)
        MessageBox::Show( "Fallo la transmición!! " );
}

private: System::Void button5_Click(System::Object^ sender,
System::EventArgs^ e) {
    int Tx=15;
    // Transmit data
    if(zgb_tx_data(Tx) == 0)
        MessageBox::Show( "Fallo la transmición!! " );
}

private: System::Void button1_Click_2(System::Object^ sender,
System::EventArgs^ e) {
    int Tx=16;
    // Transmit data
    if(zgb_tx_data(Tx) == 0)
        MessageBox::Show( "Fallo la transmición!! " );
}

private: System::Void button7_Click(System::Object^ sender,
System::EventArgs^ e) {
    int Tx=0;
    // Transmit data
    if(zgb_tx_data(Tx) == 0)
        MessageBox::Show( "Fallo la transmición!! " );
}

private: System::Void timer4_Tick(System::Object^ sender,
System::EventArgs^ e) {
    double ec;
    double dis;
    int motion;
    String ^file, ^dato;
    double action;
    int a;
```

```
int last;
last=16;
ec=0;
if(ss>7) {
    ss=1;
    if(p==0) {
        label4->Visible="False";
    }
    else{
        label4->Visible="True";
    }
    cam_obj(x,y,p);
    if(act>=6 && status==777) {
        dis=905.5*System::Math::Pow(p,-0.4162)-
10.96;

        dis=System::Math::Round(dis);
        dist->Text=dis.ToString();

        if(dis >30) {
            motion=12;
        }
        else{
            motion=13;
        }
        //if(zgb_tx_data(motion) == 0)
        //    MessageBox::Show( "Fallo la
transmición!! " );
        act=0;
    }
    if(act>=6 && status==111) {

        file=System::String::Concat("C:\\Users\\fed\\Documents\\FDMS\\FDMS\\
\\",i_f.ToString(),"\\",i_l.ToString(),".txt");
        System::IO::StreamReader ^ sr = gcnew
        System::IO::StreamReader(file);

        for(int i=0;i<i_r;i++){
            dato=sr->ReadLine();
        }
        action=System::Convert::ToDouble(dato);
        a=System::Convert::ToInt16(action);
        switch(a) {
            case 10:
                motion=11;
                label5->Text="Camina Largo";
                label7->Text=action.ToString();
                break;
            case 20:
                motion=12;
                label5->Text="Camina Medio";
                label7->Text=action.ToString();
                break;
            case 30:
```

```

        motion=13;
        label5->Text="Camina Corto";
        label7->Text=action.ToString();
        break;
    case 40:
        motion=14;
        last=14;
        label5->Text="Giro 90° a la
izquierda";

        label7->Text=action.ToString();
        break;
    case 50:
        motion=15;
        last=15;
        label5->Text="Giro 90° a la
derecha";

        label7->Text=action.ToString();
        break;
    case 60:
        motion=16;
        label5->Text="Giro 180°";
        label7->Text=action.ToString();
        break;
    case 70:
        motion=last;
        label5->Text="Especial";
        label7->Text=action.ToString();
        break;
    default:

        label5->Text="Error";
        label7->Text=action.ToString();
        break;
    }
    sr->Close();
    //if(zgb_tx_data(motion) == 0)
    //    MessageBox::Show( "Fallo la
transmisión!! " );

    act=0;
}
act++;
}

if(zgb_rx_check() == 1){
    // Get data verified
    RX = zgb_rx_data();
    //listBox1->Items->Add(RX.ToString());
    switch(ss){
        case 1:

            fkl=fk;

            ec=3579*System::Math::Pow(RX,-0.8831) -
4.526;

            fk=System::Int32(System::Math::Round(ec));
            if(fk > 130){

```

```

        fk=130;
    }
    front->Text=fk.ToString();
    i_f=fk;
    break;
case 2:
    lk=1k;
    ec=6500*System::Math::Pow(RX,-0.8235)-
14.8;

    lk=System::Int32(System::Math::Round(ec));
    if(lk > 130){
        lk=130;
    }
    left->Text=lk.ToString();

    i_l=lk;
    break;
case 3:
    rk=1k;
    ec=6500*System::Math::Pow(RX,-0.8235)-
14.8;

    rk=System::Int32(System::Math::Round(ec));
    if(rk > 130){
        rk=130;
    }
    right->Text=rk.ToString();
    i_r=rk;
    break;
case 4:
    p=RX;
    Tp->Text=p.ToString();
    break;
case 5:
    x=RX;
    Cx->Text=x.ToString();
    break;
case 6:
    y=RX;
    Cy->Text=y.ToString();

    break;
case 7:
    status=RX;
    if(status == 777)
        Status->Text="Objetivo
encontrado";

    if(status == 222)
        Status->Text="Objetivo perdido";

    if(status == 111)
        Status->Text="Explorando";

    break;
}
ss++;
```



```
        }
    }
private: System::Void listBox1_SelectedIndexChanged(System::Object^
sender, System::EventArgs^ e) {
    }
private: System::Void button7_Click_2(System::Object^ sender,
System::EventArgs^ e) {
    //Sleep(12000);
    ss=1;
    timer4->Enabled="True";
    // Transmit data
    if(zgb_tx_data(77) == 0)
        MessageBox::Show( "Fallo la transmición!! " );
    }
private: System::Void label12_Click(System::Object^ sender,
System::EventArgs^ e) {
    }
};
}
```

Código del sistema difuso

```
close all
clear all
clc

x=1:1:130;

%//////////////////SENSOR FRONTAL////////////////////////////////////
F_C = sigmf(x,[-4 31]);
F_MC= gbellmf(x,[13 20 43]);
F_M = gbellmf(x,[8 20 63]);
F_L = sigmf(x,[4 70]);

%//////////////////SENSOR IZQUIERDO////////////////////////////////////
I_C=sigmf(x,[-4 30]);
I_M=gbellmf(x,[11 20 40]);
I_L=sigmf(x,[4 50]);

% ////////////////////SENSOR DERECHA////////////////////////////////////
D_C=sigmf(x,[-4 30]);
D_M=gbellmf(x,[11 20 40]);
D_L=sigmf(x,[4 50]);

%//////////////////

% subplot(3,1,1);
% hold on
% title('Función de Membresía - Sensor Frontal')
% axis([-inf inf 0 1.2]);
% plot(x,F_C)
% plot(x,F_MC)
% plot(x,F_M)
% plot(x,F_L)
%
% subplot(3,1,2);
% hold on
% title('Función de Membresía - Sensores Laterales')
% axis([-inf inf 0 1.2]);
% plot(x,I_L)
% plot(x,I_M)
% plot(x,I_C)
%
% subplot(3,1,3);
% hold on
% title('Función de Membresía - Sensores Laterales')
% axis([-inf inf 0 1.2]);
% plot(x,D_L)
% plot(x,D_M)
% plot(x,D_C)

% //////////////////// FUNCION DE MEMBRESIA DE SALIDA //////////////////////////////////
```

```

z=0:80;
Long_F= gbellmf(z,[2 10 10]);
Medium_F= gbellmf(z,[2 10 20]);
Short_F= gbellmf(z,[2 10 30]);
I_90= gbellmf(z,[2 10 40]);
D_90= gbellmf(z,[2 10 50]);
RdP_180= gbellmf(z,[2 10 60]);
E_L= gbellmf(z,[2 10 70]);

% figure
% hold on
% title('Función de Membresia - Salida')
% axis([-inf inf 0 1.2]);
% plot(z,Long_F)
% plot(z,Medium_F)
% plot(z,Short_F)
% plot(z,I_90)
% plot(z,D_90)
% plot(z,RdP_180)
% plot(z,E_L)

% % ////////////////////////////////// CALCULO DE REGLAS IF_THEN //////////////////////////////////
[nn point_n]=size(x);

output = zeros(point_n, point_n, point_n);

for i = 1:point_n      % //////////////////////////////////SENSOR FRONTAL
    for j = 1:point_n  % //////////////////////////////////SENSOR IZQUIERDO
        for k = 1:point_n  % //////////////////////////////////SENSOR DERECHO

% ////////// DECISION-MAKING RULES////////////////////////////////////////
% ////////// LONG FORWARD //////////////////////////////////////////
        R1=F_L(i);

% ////////// MEDIUM FORWARD //////////////////////////////////////////
        R2=F_M(i);

% ////////// SHORT FORWARD //////////////////////////////////////////
        R3=min(F_MC(i),min(I_C(j),D_C(k)));
        R4=min(F_MC(i),min(I_M(j),D_M(k)));
        R5=min(F_MC(i),min(I_L(j),D_L(k)));

% ////////// GIRO IZQUIERDA //////////////////////////////////////////
        R6=min(F_MC(i),min(I_M(j),D_C(k)));
        R7=min(F_C(i),min(I_M(j),D_C(k)));

        R8=min(F_MC(i),min(I_L(j),D_C(k)));
        R9=min(F_C(i),min(I_L(j),D_C(k)));

        R10=min(F_MC(i),min(I_L(j),D_M(k)));
        R11=min(F_C(i),min(I_L(j),D_M(k)));

```

```
% ////////// GIRO DERECHA ////////////////////////////////////////
R12=min(F_MC(i),min(I_C(j),D_M(k)));
R13=min(F_C(i),min(I_C(j),D_M(k)));

R14=min(F_MC(i),min(I_C(j),D_L(k)));
R15=min(F_C(i),min(I_C(j),D_L(k)));

R16=min(F_MC(i),min(I_M(j),D_L(k)));
R17=min(F_C(i),min(I_M(j),D_L(k)));

% ////////// GIRO 180° ////////////////////////////////////////
R18=min(F_C(i),min(I_C(j),D_C(k)));

% ////////// ESPECIAL ////////////////////////////////////////
R19=min(F_C(i),min(I_M(j),D_M(k)));
R20=min(F_C(i),min(I_L(j),D_L(k)));

%//////////////////////////////////////

cut_mf(1,:) = min(R1, Long_F);
cut_mf(2,:) = min(R2, Medium_F);
cut_mf(3,:) = min(R3, Short_F);
cut_mf(4,:) = min(R4, Short_F);
cut_mf(5,:) = min(R5, Short_F);
cut_mf(6,:) = min(R6, I_90);
cut_mf(7,:) = min(R7, I_90);
cut_mf(8,:) = min(R8, I_90);
cut_mf(9,:) = min(R9, I_90);
cut_mf(10,:) = min(R10, I_90);
cut_mf(11,:) = min(R11, I_90);
cut_mf(12,:) = min(R12, D_90);
cut_mf(13,:) = min(R13, D_90);
cut_mf(14,:) = min(R14, D_90);
cut_mf(15,:) = min(R15, D_90);
cut_mf(16,:) = min(R16, D_90);
cut_mf(17,:) = min(R17, D_90);
cut_mf(18,:) = min(R18, RdP_180);
cut_mf(19,:) = min(R19, E_L);
cut_mf(20,:) = min(R20, E_L);

overall_out_mf = max(cut_mf);
output(j,k,i) = defuzzy(z, overall_out_mf,3);

end
end
end

% output2 = zeros(point_n, point_n, point_n);
% d=zeros(1,7);
% for i = 1:point_n      % ////////////////////////////////////SENSOR FRONTAL
%   for j = 1:point_n    % ////////////////////////////////////SENSOR IZQUIERDO
%     a=num2str(i);
```

```
% b=strcat('C:\Users\fed\Documents\FDMS\FDMS\',a);
% a=num2str(j);
% b=strcat(b,'\');
% c=strcat(b,a);
% c=strcat(c, '.txt');
% file=fopen(c,'wt','n');
% for k = 1:point_n      % ////////////////////////////////////SENSOR DERECHO
%   d(1,1)=sqrt((output(j,k,i)-10)^2);
%   d(1,2)=sqrt((output(j,k,i)-20)^2);
%   d(1,3)=sqrt((output(j,k,i)-30)^2);
%   d(1,4)=sqrt((output(j,k,i)-40)^2);
%   d(1,5)=sqrt((output(j,k,i)-50)^2);
%   d(1,6)=sqrt((output(j,k,i)-60)^2);
%   d(1,7)=sqrt((output(j,k,i)-70)^2);
%   m=d(1,1);
%   im=1;
%   for l=2:7
%     if m > d(1,l)
%       m=d(1,l);
%       im=l;
%     end
%   end
%   switch im
%     case 1
%       output2(j,k,i)=10;
%     case 2
%       output2(j,k,i)=20;
%     case 3
%       output2(j,k,i)=30;
%     case 4
%       output2(j,k,i)=40;
%     case 5
%       output2(j,k,i)=50;
%     case 6
%       output2(j,k,i)=60;
%     case 7
%       output2(j,k,i)=70;
%   end
%   fprintf(file, '%d \n', output2(j,k,i));
% end
% fclose(file);
% end
% end
```