INSTITUTO POLITÉCNICO NACIONAL

Centro de investigación en computación

Laboratorio de robótica y mecatrónica

"Control de un robot humanoide mediante redes neuronales pulsantes para la manipulación de objetos"

TESIS

Que para obtener el grado de
Maestría en Ciencias de la Computación:

Presenta

Brauer Luna Polo Mauricio

Directores

Dr. Juan Humberto Sossa Azuela

Dra. Elsa Rubio Espino

México, D.F., Junio de 2013

SIP-14 bis

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 12:00 horas del día 28 del mes de enero 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:

"Control de un robot humanoide mediante redes neuronales pulsantes para la manipulación de objetos"

Presentada por el alumno:

BRAUER	LUNA POLO	MAURICIO ALBERTO
Apellido paterno	Apellido materno	Nombre(s)

Con registro: B 1 1 0 8 4 6

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. Carlos Fernando Aguilar Ibáñez

Dr. Marco Antonio Moreno Armendáriz

Dr. Francisco Hiram Calvo Castro

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. José Alfonso Villa Vargas

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION EN COMPUTACION
DIRECCION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 13 del mes Junio del año 2013, el (la) que suscribe Mauricio Alberto Brauer Luna Polo alumno (a) del Programa de Maestría en Ciencias de la Computación con número de registro B110846, adscrito a Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Juan Humberto Sossa Azuela y Dra. Elsa Rubio Espino y cede los derechos del trabajo intitulado "Control de un robot humanoide mediante redes neuronales pulsantes para la manipulación de objetos", 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 pbrauer_b11@cic.ipn.mx. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Mauricio Alberto Brauer Luna Polo.

Para Lita

Agradecimientos

A Dios.

Por otorgarme la vida y darme la fortaleza para alcanzar mis objetivos. Por hacerse presente en todos los aspectos de mi vida y mostrarme el camino de la verdad.

A mi madre.

Por haber sido la mejor madre y padre que pude haber tenido, y por haberme dado su apoyo incondicional día con día. Por soportar los momentos difíciles y representar el soporte más importante de mi vida.

A mi abuelita.

Por permanecer en mi memoria y en mi espíritu todos los días. Por ofrecerme apoyo vital durante toda su vida y después de ella. Por significar uno de los castillos durante mi crecimiento.

A mi novia.

Por estar siempre a mi lado y contagiarme de todas sus ganas de lograr las metas. Por conservar su actitud positiva en cada momento y ayudarme en todo momento que la necesité...

A mis directores.

Por brindarme su paciencia, conocimientos y apoyo para lograr este objetivo, a la vez que me dieron la oportunidad de compartir sus experiencias que me enriquecieron como estudiante y persona.

A CONACyT y al Instituto Politécnico Nacional

Por los apoyos económicos para la realización de este trabajo de investigación. Este trabajo de investigación fue realizado en el marco de los proyectos con número de registro: SIP 20121311, SIP20131182 y CONACYT 155014.

Resumen

El presente trabajo muestra una metodología para clasificar las acciones de un robot humanoide, mediante algoritmos verosímiles de aprendizaje automático tales como las redes neuronales pulsantes.

Mediante dicha clasificación, el robot humanoide ejecuta la tarea de desplazarse, de tomar un objeto y de transportarlo hasta una posición meta predefinida, obedeciendo a un comando verbal.

El sistema cuenta con muy poca información acerca de cuáles acciones se deben tomar en diferentes circunstancias y hace uso de la capacidad de generalización de las neuronas pulsantes, de la visión por computadora y del control difuso para poder ejecutar la tarea de forma exitosa.

Abstract

This thesis shows a methodology for a humanoid robot action classification by using plausible machine learning methods such as spiking neural networks.

By identifying the action, the robot performs the task of navigating, taking an object and placing it in a pre-defined target position.

The system has a small amount of information about which action should be taken under different circumstances and it uses the generalization capability of the spiking neurons, computing vision algorithms and fuzzy logic in order to perform the task successfully.

Contenido

Capítulo 1. Introducción ... 3

1.1 Antecedentes ... 4

1.2 Planteamiento del problema ... 6

1.3 Justificación ... 7

1.4 Objetivo general ... 7

1.5 Objetivos específicos ... 8

1.6 Consideraciones iniciales ... 8

1.7 Organización de la tesis ... 9

Capítulo 2. Estado del arte ... 10

2.1 Pattern recognition using spiking neurons and firing rates ... 10

2.2 Object Grasping with the NAO ... 11

2.3 Localización y manipulación de objetos mediante un robot móvil Khepera II auxiliado de una cámara y un elemento prensil ... 12

2.4 Natural Methods for robot task learning: instructive demonstrations, generalization and practice ... 13

2.5 iSpike: A Spiking Neural Interface for the iCub robot ... 14

Capítulo 3. Marco teórico ... 15

3.1 Aprendizaje automático ... 15

3.1.1 Aprendizaje supervisado ... 16

3.1.2 Aprendizaje por demostración ... 17

3.1.3 Generalización ... 17

3.1.4 Redes neuronales ... 19

3.1.5 Redes neuronales pulsantes ... 20

3.1.5.1 Neuronas pulsantes biológicas ... 21

3.1.5.2 Tipos de neuronas pulsantes ... 22

3.1.5.3 Modelo de Izhikevich. ... 24

3.1.5.4 Pulsos rápidos ... 26

3.1.5.5 Cálculo de la corriente de entrada y frecuencia de salida. 29
3.1.5.6. Clasificación. 30
3.2 Visión por computadora 33
3.2.1 Reconocimiento del color 34
3.2.2 Regiones crecientes 35
3.2.3. Extracción de rasgos 35
3.3 Lógica difusa 37
3.3.1. Conjuntos difusos 37
3.3.2 Variable lingüística 37
3.3.3. Funciones de membresía 38
3.3.4. Base de reglas difusas 39
3.3.5. Modelo de inferencia de Mamdani 40
3.4 BIOLOID Premium Kit 41
3.4.1 Dynamixel AX-12A 43
3.4.2 CM-510 44
3.4.3 USB2Dynamixel 45
3.4.4 Zig-100/110 46
3.4.5 Sensor de distancia 47
3.4.6 Sensor de giro 48
3.4.7 Roboplus 49
3.4.7.1. Roboplus Task 49
3.4.7.2. Roboplus Motion 50
3.4.7.3. Roboplus Manager 50
3.4.8 Cámara HAVIMO 2.0 50
3.5 Java Speech 53
Capítulo 4. Metodología 56
4.1 Descripción del sistema 56
4.2 Arquitectura del sistema 56

4.3 Módulo: Robot humanoide 58
4.3.1 Rasgos 58
4.3.2 Acciones del robot 60
4.4 Módulo: Computadora. 61
4.4.1 Neurona pulsante 61
4.4.1.1 Conjunto de entrenamiento. 61
4.4.1.2 Entrenamiento de la red pulsante. 63
4.4.1.3 Clasificador euclidiano 68
4.4.2 Sistema difuso. 69
4.4.3. Java Speech 72
Capítulo 5. Resultados 74
5.1 Experimento: Localizar objeto 74
5.2 Experimento: Desplazarse y tomar objeto 76
5.3 Experimento: Tomar objeto y transportarlo 84
Capítulo 6. Conclusiones y trabajo futuro 91
6.1 Conclusiones. 91
6.2 Trabajo futuro. 92
Apéndice A 94
Apéndice B 99
Apéndice C 103
Referencias 114

Índice de figuras

Figura 2.1.1. Conjunto de imágenes utilizadas en la tesis: “Pattern recognition using spiking neurons and firing rates” 10

Figura 2.2.1. Geometría del NAO 11

Figura 2.3.1 Robot Khepera manipulando objetos 12

Figura 2.4.1 Aprendizaje del robot Piooner 2-DX 13

Figura 2.5.1. Biblioteca iSpike y robot iCub 14

Figura 3.1.1.1 Diagrama de flujo aprendizaje supervisado 16

Figura 3.1.4.1. Neurona artificial 20

Figura 3.1.5.1.1. Partes de la neurona. 22

Figura 3.1.5.2.1. Tipos de neuronas pulsantes. 23

Figura 3.1.5.3 Modelo de Izhikevich. 24

Figura 3.1.5.3.1. Comparativa entre modelos de neuronas pulsantes 25

Figura 3.1.5.4.1 Pulsos rápidos. 27

Figura 3.1.5.4.1. Ejemplo del tren de pulsos generado por la neurona de Izhikevich 28

Figura 3.1.5.5.1. Conversión del vector de rasgos en corriente. 29

Figura 3.1.5.5.2. Ejemplo del cálculo de frecuencias con base en un vector de rasgos 30

Figura 3.1.5.6.1. Hiperplano que maximiza la separación entre las clases. 30

Figura 3.1.5.7.1. Topología utilizada para probar neurona de segunda generación 107

Figura 3.2.1. Componentes de un sistema reconocedor de objetos 33

Figura 3.3.3.1 Función de membresía triangular 38

Figura 3.3.3.2 Función de membresía trapezoidal 39

Figura 3.3.5.1. Modelo de inferencia tipo Mamdami 40

Figura 3.4.1. Tipos de humanoide del BIOLOID Premium Kit 42

Figura 3.4.2. Principales componentes del BIOLOID Premium Kit 43

Figura 3.4.1.1. Dyamixel AX-12A 43

Figura 3.4.2.1. Controlador CM510 44

Figura 3.4.3.1. a) Control de dynamixeles usando PC b) Controlando por del puerto serial c) Comunicación inalámbrica 46

Figura 3.4.4.2. Zig2Serial 47

Figura 3.4.4.3. Conexión entre Zig2Serial y Zig-100 47

Figura 3.4.4.4. Conexión entre el Zig2Serial y el USB2Dynamixel 47

Figura 3.4.5.2. Distancia vs valor arrojado por el sensor de distancia 48

Figura 3.4.6.1. Sensor de giro 48

Figura 3.4.8.1. Cámara Havimo 51

Figura 3.4.8.2. Conexión entre la cámara Havimo y la computadora 52

Figura 3.4.8.3. HAVIMO_GUI 52

Figura 3.4.8.4. Datos accesibles desde un programa en RoboPlus Task. 53

Figura 3.5.1. Partes del API de JavaSpeech 54

Figura 4.1.1. Módulos del sistema. 56

Figura 4.2.1. Arquitectura general del sistema 57

Figura 4.3.1.1. Rasgos seleccionados 58

Figura 4.3.1.2. División de los rasgos obtenidos por la computadora. 59

Figura 4.3.2.1. Posibles acciones del robot humanoide. 60

Figura 4.4.1 Sub-módulos de la computadora. 61

Figura 4.4.1.1.1. Construcción del conjunto de entrenamiento. a) Girar derecha. b) Tomar objeto. c) Avanzar. d) Girar izquierda. 62

Figura 4.4.1.1.2 Interfaz para capturar datos 63

Figura 4.4.1.2.1. Diagrama de flujo del algoritmo de entrenamiento 66

Figura 4.4.1.2.3. Separación gráfica entre las clases. 68

Figura 4.4.1.3.1. Clasificador euclidiano 69

Figura 4.4.5.1. Conjuntos difusos para la acción de Avanzar. a) Conjuntos difusos antecedentes. b) Conjuntos difusos consecuentes. 70

Figura 4.4.5.1. Relación de píxeles con segundos. 70

Figura 4.4.5.3. Conjuntos difusos para las acciones “Girar izquierda” y “Girar derecha”. a) Conjuntos difusos antecedentes. b) Conjuntos difusos consecuentes. 71

Figura 5.1.1. Localización: Configuración 1. 74

Figura 5.1.2. Localización: Configuración 2. 75

Figura 5.1.3. Localización: Configuración 3. 75

Figura 5.1.4. Localización: Configuración 4. 75

Figura 5.1.5. Casos de éxito donde se localizó el objeto en cada una de las diferentes configuraciones descritas 76

Figura 5.2.1. Tomar objeto: Configuración 1. 77

Figura 5.2.2. Tomar objeto: Configuración 2. 79

Figura 5.2.3. Tomar objeto: Configuración 3. 80

Figura 5.2.4. Tomar objeto: Configuración 4. 82

Figura 5.2.5. Casos de éxito donde se tomó el objeto en cada una de las diferentes configuraciones descritas 83

Figura 5.2.6. Tiempo promedio que tardó el robot en tomar el objeto en cada configuración. 83

Figura 5.3.1. Transportar objeto: Configuración 1. 85

Figura 5.3.2. Transportar objeto: Configuración 2. 86

Figura 5.3.3. Transportar objeto: Configuración 3. 87

Figura 5.3.4. Transportar objeto: Configuración 4. 88

Figura 5.3.5. Casos de éxito donde se transportó el objeto en cada una de las diferentes configuraciones descritas 89

Figura 5.3.6. Tiempo promedio que tardó el robot en transportar el objeto en cada configuración. 89

Figura A.2.1. Acceso a la cámara HAVIMO 2.0. (a) Configuración de la cámara. (b) Función Get Bounding Box. (c) Acceso a los datos de la cámara. 97

Figura A.2.2. Envío y recepción de datos. 98

Figura A.2.3. Ejecución de la acción. 98

Figura B.2.1. Salida del sistema difuso con 817 píxeles de entrada. 101

Figura B.2.3. Salida del sistema difuso con el motor en 684 grados. 102

Figura C.1 Medición de desempeño: variación del número de rasgos en 2 clases 103

Figura C.2 Medición de desempeño: variación del número de clases con 2 rasgos. 104

Figura C.3 Medición de desempeño: variación del número de rasgos y clases 104

Índice de tablas

Tabla 3.1.3.1. Ventajas y desventajas de algoritmos de aprendizaje supervisado 18

Tabla 3.1.5.4.1. Ejemplo numérico de la generación de un pulso..... 28

Tabla 3.1.5.6.1 Conjunto de entrenamiento para el ejemplo numérico de un problema de clasificación. 31

Tabla 3.1.5.6.2. Frecuencia generada por el conjunto de entrenamiento para el ejemplo numérico de un problema de clasificación. 31

Tabla 3.1.5.6.3. Clasificación de los vectores de prueba para un ejemplo numérico. 31

Tabla 3.4.1. Comparativa entre los KITS de BIOLOID..... 41

Tabla 3.4.2. Comparativa de las características de los humanoides 42

Tabla 3.4.1.1. Principales características del dynamixel AX-12A 44

Tabla 3.4.2.1. Características del CM-510. 45

Tabla 3.4.8.1. Principales características de la cámara HAVIMO 2.0..... 51

Tabla 4.4.1.1.1. Conjunto de entrenamiento de la red neuronal pulsante. 63

Tabla 4.4.1.2.1. Ejemplo del cálculo de la distancia entre clases. 64

Tabla 4.4.1.2.2. Evolución de los pesos sinápticos y la distancia entre clases. 67

Tabla 4.4.1.2.3. Frecuencias generadas por el conjunto de entrenamiento después del ajuste pesos sinápticos. 67

Capítulo 1

Introducción

El campo de la robótica ha evolucionado a grandes pasos y no es sorpresa que los robots se incorporen a nuestra vida cotidiana en el corto plazo. Dado que los robots pueden ejecutar tareas durante largos períodos de tiempo sin necesidad de comer o dormir, dentro de pocos años los robots serán una herramienta de apoyo a las actividades domésticas, de vigilancia, industriales e incluso de exploraciones espaciales.

A pesar de que algunos robots que realizan tareas útiles se han puesto a la venta en el mercado tal como los robots limpiadores como el Roomba y robots cortadores de césped [20] [21], sus capacidades son muy limitadas dado que están totalmente enfocados a realizar una sola tarea.

Gran parte de la investigación actual se está llevando en robots de propósito general, estimulada principalmente por competencias como RoboCup@home [22], organizada desde 2006. Este tipo de competencias, tiene como objetivo construir robots autónomos de servicio doméstico para asistir a usuarios en ambientes de la casa.

Las competencias han demostrado que una de las características principales que requiere tener un robot, es la capacidad de manipular objetos, dicha tarea tiene como objetivo ejercer fuerza sobre un objeto para tomarlo, sostenerlo y transportarlo hasta un lugar deseado. La manipulación de objetos resulta una tarea crucial para los robots de servicio, los cuales, tienen como único propósito el de ayudar y apoyar al ser humano en sus tareas cotidianas [9].

Entre las aplicaciones donde se está realizando investigación en robots manipuladores se destacan las siguientes:

- Tareas del hogar
- Apoyo a personas incapacitadas o de edad avanzada
- Rescate de seres humanos

- Transporte de bombas
- Exploraciones espaciales

Dado que los robots de servicio requieren interactuar con personas en ambientes diseñados por el ser humano, no es de extrañarse que este tipo de robots tengan una forma humana que les permita manejar herramientas y ejecutar tareas de forma versátil [8]. Por otra parte, se espera que los robots de servicio sean capaces de aprender a ejecutar nuevas tareas en situaciones desconocidas, ya que el mundo real es dinámico y muy complejo; entonces los robots deben ser capaces de adaptarse al cambio y poder ejecutar tareas de forma satisfactoria de acuerdo a las necesidades del operador.

1.1 Antecedentes

Los robots humanoides cuentan con un cuerpo de figura antropomórfica y sentidos parecidos a los de un ser humano. Las investigaciones realizadas en este tipo de robots están motivadas por la visión de crear una nueva herramienta: robots que trabajen con seres humanos o en el mismo ambiente diseñado para nuestras necesidades.

El cuerpo humano está bien adecuado para actuar en nuestros ambientes cotidianos. Escaleras, perillas de puertas, herramientas, entre otras han sido diseñadas por seres humanos, y por lo tanto, un cuerpo con características físicas humanas tiene ventaja en este tipo de ambientes.

Aunque robots industriales especializados son empleados exitosamente en producciones industriales en masa, los robots humanoides requieren un enfoque diferente [10].

Los robots humanoides de servicio requieren poseer la capacidad de que un ser humano, sin experiencia en el ámbito de la programación, sea capaz de integrar movimientos, habilidades de percepción o estrategias para solucionar nuevos problemas; es por ello, que surge la necesidad de integrar algoritmos de aprendizaje de máquinas a este tipo de herramientas [18].

El aprendizaje de máquinas o aprendizaje automático consiste en tener sistemas de propósito general que se puedan adaptar a las circunstancias; en lugar de escribir explícitamente un programa diferente para una circunstancia en especial. El aprendizaje es necesario en casos donde nosotros no podemos escribir directamente el programa en la computadora para resolver el problema, por ejemplo, cuando el problema a ser resuelto cambia con el tiempo o depende del medio ambiente [17].

Dentro del campo de la robótica se han aplicado diferentes enfoques del aprendizaje de máquinas, sin embargo, el aprendizaje supervisado ha demostrado tener muy buenos resultados.

El aprendizaje supervisado consiste en inferir una función dado un conjunto de datos de entrenamiento, el cual, consiste en pares de datos entrada-salida. Por lo general la entrada es un vector. El algoritmo de aprendizaje supervisado analiza los datos de entrenamiento y produce una función, la cual, es capaz de predecir la salida apropiada para cualquier dato de entrada válido [16].

Dentro del aprendizaje supervisado existen diferentes técnicas y algoritmos desarrollados para la aproximación de funciones, entre ellas destacan las redes neuronales artificiales.

A diferencia de la inteligencia artificial clásica, la nueva inteligencia artificial señala que la investigación práctica del problema mente-cuerpo no puede ser abordada considerando la mente y el cuerpo como entidades separadas, es decir, el paradigma refiere que el cuerpo, la mente y el ambiente están conectados [6].

Es por ello, que el combinar robots humanoides con algoritmos inspirados en la estructura y funcionamiento del cerebro humano cobra sentido.

Las redes neuronales artificiales tienen su origen en la necesidad de formalizar el pensamiento humano. Son una aproximación para modelar las capacidades del procesamiento de información de sistemas nerviosos.

Después de 1943 cuando apareció el primer modelo simple de redes neuronales propuesto por McCulloch y Pitts comenzaron a aparecer diferentes variaciones del modelo y algoritmos para realizar el aprendizaje de las redes, entre ellos destacan el

perceptrón de Rosenblatt en 1958, continuado por el ajuste de pesos sinápticos desarrollado por Widrow y Hoff en 1958. Parker desarrolló el algoritmo de retropropagación en 1985, con el cual, se demostró que las redes neuronales podían resolver problemas no lineales [12].

Durante los últimos 25 años, después de desarrollarse el algoritmo de retropropagación, diferentes autores han propuesto modelos, los cuales, recaen entre las llamadas primera y segunda generación de redes neuronales; estas redes arrojan una salida binaria.

Sin embargo, actualmente gran parte de la investigación actual sobre redes neuronales está enfocada en realizar modelos que se aproximen más al funcionamiento de las neuronas reales y además sean computacionalmente eficientes. Con el objetivo de satisfacer esta necesidad surgieron las redes neuronales pulsantes, las cuales, recaen en la tercera generación de redes neuronales, ya que incrementan el nivel de realismo biológico, esto se debe a que tienen una salida de pulsos y no binaria; además incorporan información espacio-temporal durante la comunicación y los cálculos, tal como las neuronas reales lo hacen [13].

Las redes neuronales pulsantes han sido utilizadas actualmente en áreas tales como biología, reconocimiento de patrones, análisis de imágenes médicas, robótica entre otras [1]. Es por ello, que resulta comprensible pensar que este tipo de redes neuronales se pueden combinar con los robots humanoides y así dotarlos de nuevas capacidades que les permitan tener un comportamiento que se asemeje más al nuestro y abordar el problema cuerpo-mente desde una perspectiva más novedosa y cercana a la realidad.

1.2 Planteamiento del problema

Las competencias de RoboCup@Home han demostrado que los robots de servicio deben ser capaces de poder ejecutar tareas de navegación y manipulación de objetos a partir de una pequeña cantidad de información. Los robots deben poseer herramientas que les permitan generalizar ésta información y ejecutar la tarea de forma exitosa en escenarios desconocidos.

Para lograr esto los robots deben tener la capacidad de percibir su entorno mediante diferentes sensores y cámaras pero además poseer mecanismos de aprendizaje que les permita poder adquirir una nueva habilidad mediante la enseñanza de un ser humano.

La presente tesis busca hacer una integración de herramientas que permitan controlar a un robot, dotado con sensores y un sistema de visión, con el propósito de aprender a ejecutar una tarea de navegación y manipulación de objetos con una cantidad mínima de información dada por el usuario.

1.3 Justificación

Los robots móviles manipuladores de objetos significan un gran apoyo en actividades que el ser humano no puede hacer, actividades que son peligrosas o tareas que representan gasto de tiempo y esfuerzo. Desde actividades del hogar tales como cocinar, limpiar o trapear hasta actividades de exploración espacial, la manipulación de objetos representa una tarea crucial. El dotar a un robot con la capacidad de aprender a ejecutar dichas tareas de manipulación permitirá otorgarles la versatilidad necesaria para incorporarlos a nuestra vida cotidiana.

1.4 Objetivo general

Dado un robot humanoide H de 18 grados de libertad y equipado con un sistema de visión, un objeto O con forma geométrica cilíndrica, un solo color, con peso y dimensiones proporcionales a la capacidad de H y una posición meta G representada por un objeto con forma cilíndrica y de color diferente a O; controlar a H de forma que en respuesta a un comando verbal:

1) Busque y localice a O dentro del espacio de trabajo.
2) Tome decisiones para ubicarse hasta la posición donde se encuentra O.
3) Se posicione con respecto a O, de tal manera, que sea capaz de tomarlo, sostenerlo y transportarlo.
4) Transportar y llevar O hasta la posición de G.

1.5 Objetivos específicos

- Especializarse en la tecnología del robot humanoide BIOLOID Premium Kit y el sistema de visión HAVIMO 2.0.
- Diseñar e implementar una red neuronal pulsante que permita clasificar las acciones del robot humanoide.
- Implementar un sistema difuso que permita controlar el tiempo que las acciones serán ejecutadas.
- Crear un framework que permita generalizar el algoritmo utilizado en la presente tesis.

1.6 Consideraciones iniciales

Las consideraciones iniciales que se tomaron en cuenta para realizar el desarrollo del sistema.

- El entorno de trabajo será un área de 100 cm $*$ 100 cm.
- No deberá existir ningún objeto de color en el área, con excepción de los objetos a transportar.
- La superficie será plana sin irregularidades.
- La iluminación del entorno será constante.
- El objeto a desplazar contará con las siguiente características:
    - Forma cilíndrica.
    - Un solo color liso.
    - El peso es 50 gr.
    - Las dimensiones del objeto son:
        - 10 cm de diámetro.
        - 20 cm de alto.
- La posición meta será representada por otro objeto con las siguientes características:
    - Forma cilíndrica.
    - Un solo color liso.
    - Las dimensiones del objeto son:
        - 25 cm de diámetro.

- 40 cm de alto.

1.7 Organización de la tesis

Capítulo 2: Se presenta el estado del arte sobre los trabajos más importantes y más recientes en relación a robots móviles manipuladores y las redes neuronales pulsantes.

Capítulo 3: Se hace una descripción de las herramientas utilizadas.

Capítulo 4: Se detallan la metodología seguida durante la presente tesis.

Capítulo 5: Se detallan los resultados obtenidos en la tesis.

Capítulo 6: En este capítulo se muestran las conclusiones alcanzadas, así como el trabajo futuro propuesto.

Capítulo 2

Estado del arte

Durante este capítulo se detallan los trabajos más relevantes acerca del desarrollo de sistemas que involucran redes neuronales pulsantes y robots humanoides, además se describe la metodología, las herramientas y los algoritmos utilizados en cada uno de ellos.

2.1 Pattern recognition using spiking neurons and firing rates

En este trabajo realizado por Roberto A. Vázquez en la Universidad de La Salle (2010) [15], se describe el uso de neuronas pulsantes y la tasa de pulsos que generan para resolver un problema de clasificación de objetos. La figura 2.1.1 muestra el conjunto de imágenes utilizada en el trabajo.

Figura 2.1.1. Conjunto de imágenes utilizadas en la tesis: "Pattern recognition using spiking neurons and firing rates".

En el artículo el autor hace uso de una neurona pulsante basada en el modelo de Izhikevich con el objetivo de generar tasas de pulsos, las cuales, deben ser similares para la misma clase pero, para clases diferentes, debe ser los suficientemente diferente para discriminar entre las clases.

El sistema hace el análisis de las imágenes y arroja los 7 invariantes Hu, los cuales, son multiplicados por los pesos sinápticos de la neurona pulsante y se obtiene un tren de picos a la salida, el cual, es dividido entre una ventana de tiempo de 1000 ms, con el propósito de obtener la tasa de pulsos.

El autor propone un método basado en el algoritmo de evolución diferencial para realizar el ajuste automático de los pesos sinápticos y demuestra que la neurona

pulsante tiene mejores resultados que las redes neuronales de propagación hacia adelante en problemas no lineales.

2.2 Object Grasping with the NAO

El trabajo desarrollado por Egbert van der Wal [24] en University of Groningen, en Holanda (2012), aborda el problema de aprendizaje en un robot humanoide NAO, con el objetivo de mover un brazo en la trayectoria apropiada para agarrar un objeto. La geometría del robot humanoide NAO es mostrada en la figura 2.2.1.

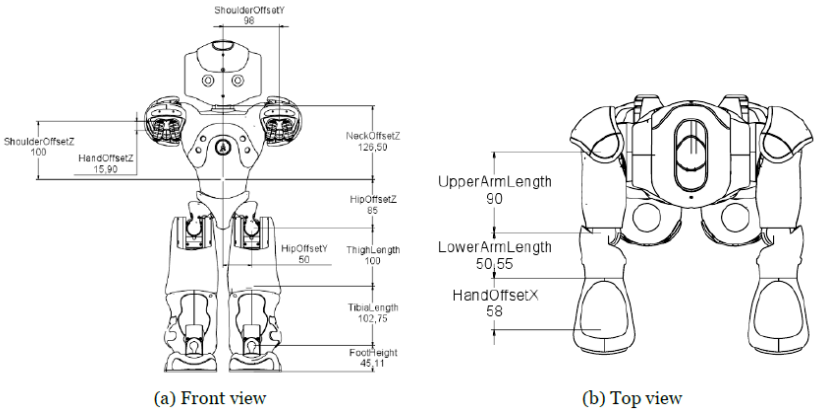


Figura 2.2.1. Geometría del NAO.

El autor hace la evaluación de tres algoritmos: aprendizaje por demostración haciendo uso de una red neuronal artificial, algoritmo de los vecinos más cercanos y aprendizaje por refuerzo.

El autor hace uso de ejemplos activos en el aprendizaje por demostración, lo cual, no tuvo los resultados esperados con la red neuronal artificial. El problema con los ejemplos activos, es que deben ser demasiados para que el sistema se comporte de manera adecuada.

El algoritmo de los vecinos más cercanos muestra que el usar la interpolación entre los datos para encontrar la acción correcta, converge en un desempeño adecuado, ésta es una aproximación muy cercana al uso de las redes neuronales pulsantes.

2.3 Localización y manipulación de objetos mediante un robot móvil Khepera II auxiliado de una cámara y un elemento prensil

En esta tesis desarrollada por Gabriel Omar Huerta Moreno en el CIC-IPN (2009) [25], en ella se describe una metodología dirigida hacia el control del robot Khepera mediante lógica difusa y visión por computadora.

La figura 2.3.1 muestra el sistema en cuestión.

Figura 2.3.1 Robot Khepera manipulando objetos.

El sistema es capaz de hacer una clasificación exitosa entre cuerpos geométricos de forma sencilla tales como un cubo y un objeto de figura cilíndrica, ya que introduce un vector de invariantes de Hu, a una red neuronal multicapa, la cual, le permite hacer una distinción entre su geometría.

El sistema hace uso de un sistema difuso tipo Mamdani, donde los conjuntos difusos antecedentes están dados por la orientación y la distancia con respecto al objeto, mientras que los consecuentes son las velocidades de la rueda izquierda y la derecha del robot.

El autor reporta entre un 70% y 80 % de eficiencia del sistema al ejecutar diferentes experimentos que consistieron en buscar, tomar los objetos y colocarlos en lugares indicados por el usuario.

2.4 Natural Methods for robot task learning: instructive demonstrations, generalization and practice

En este trabajo realizado en University of Southern California por Monica Nicolescu y Maja Mataric (2003) [27], se hace uso del robot móvil Pioneer 2-DX, el cual, cuenta con 2 anillos de sonares (8 frontales y 8 traseros) una cámara y un objeto prensil.

El trabajo busca el que el robot aprenda a localizar, tomar y soltar un objeto en una posición meta mediante el uso de comandos de voz tal como se muestra en la figura 2.4.1.

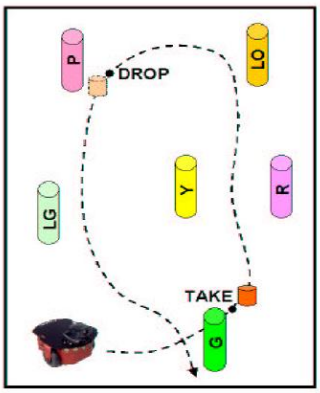


Figura 2.4.1 Aprendizaje del robot Piooner 2-DX.

Con el objetivo de realizar el aprendizaje del robot se hace uso de demostraciones experimentadas, las cuales, implican que el robot participe activamente en la demostración hecha por el maestro y vive la experiencia a través de sus sensores.

El trabajo no busca reproducir las trayectorias exactas del maestro, en cambio, busca hacer uso de la generalización para que el robot relacione un estado con una acción.

El autor hace énfasis en la necesidad del robot por generalizar el conocimiento a partir del conjunto de datos inicial. Dado que hace uso de una representación tipo gráfico para la información recopilada por el robot el autor utiliza un método de generalización basado en topologías.

2.5 iSpike: A Spiking Neural Interface for the iCub robot

En esta tesis realizada por David Gamez, A.K Fidjeland y E. Lazdins en University of Sussex (2012) [28], se presenta el desarrollo de una biblioteca en C++ que funge como interfaz entre simuladores de redes neuronales pulsantes y el robot humanoide iCub.

La biblioteca codifica la información de los sensores del motor en forma de pulsos y los introduce en un simulador de redes neuronales pulsantes, el cual arroja como salida otra serie de pulsos que son convertidos en comandos para los motores y el control del robot.

La figura 2.5.1 muestra el papel que juega la librería iSpike en el control del robot iCub.

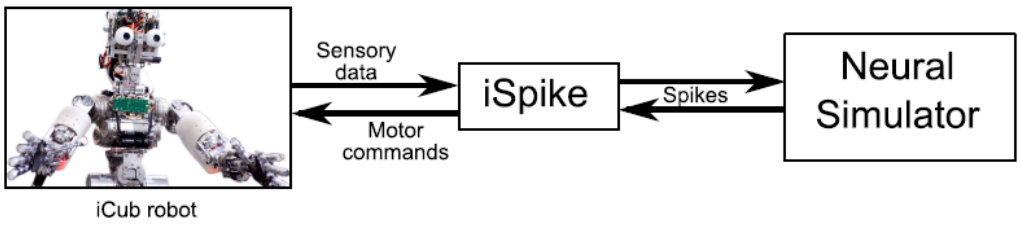


Figura 2.5.1. Biblioteca iSpike y robot iCub.

En este trabajo se realizó una integración del simulador neuronal NeMo y su interfaz gráfica SpikeStream, el cual, provee visualización 3D de la red neuronal pulsante, sin embargo, el sistema es compatible con cualquier otro simulador con una interfaz escrita en el lenguaje C++.

El sistema busca hacer uso de los robots humanoides para modelar ciertos aspectos del cerebro y explorar la inteligencia que los sistemas pueden alcanzar haciendo uso de de redes neuronales pulsantes.

Capítulo 3

Marco teórico

En este capítulo se detallan las herramientas utilizadas en la tesis. Primeramente se realiza una descripción sobre el aprendizaje de máquinas y las redes neuronales pulsantes, seguido de los principios de lógica difusa y finalmente se muestran las características del robot humanoide BIOLOID y el sistema de visión HAVIMO 2.0.

3.1 Aprendizaje automático

El aprendizaje automático consiste en diseñar programas que pueden aprender reglas a partir de un conjunto de datos [14]. El aprendizaje es necesario en casos donde nosotros no podemos escribir directamente el programa en la computadora para resolver el problema [17]. Es por ello, que el aprendizaje automático está relacionado con los robots de servicio, ya que serán usados por personas que no necesariamente tengan conocimientos sobre programación [18].

Los algoritmos de aprendizaje de máquinas pueden ser organizados en una taxonomía basada en la salida del algoritmo y el tipo de entrada durante el entrenamiento de la máquina. Los algoritmos son los siguientes:

- Aprendizaje supervisado
- Aprendizaje no supervisado
- Aprendizaje por refuerzo

El aprendizaje supervisado tiene algunas ventajas con respecto al aprendizaje por refuerzo y al no supervisado en el campo de los robots de servicio, por ejemplo, el aprendizaje supervisado, a diferencia del aprendizaje reforzado, no requiere conocimientos del operador sobre cómo definir una función de recompensa. Por otra parte, el llevar a cabo un aprendizaje no supervisado en el campo de los robots de servicio, no siempre resulta factible ya que los algoritmos requieren exponer al robot al mismo problema en varias ocasiones para que sea capaz de reconocer patrones y etiquetar las salidas [26]. En la presente tesis se hará uso del aprendizaje supervisado ya que no requiere que el usuario sea un experto y además es intuitivo.

3.1.1 Aprendizaje supervisado

El aprendizaje supervisado consiste en inferir una función dado un conjunto de datos de entrenamiento, el cual, consiste en pares de datos entrada-salida. Por lo general la entrada es un vector. El algoritmo de aprendizaje supervisado analiza los datos de entrenamiento y produce una función, la cual, es capaz de predecir la salida apropiada para cualquier dato de entrada válido [16].

La figura 3.1.1.1 muestra el proceso de aplicar el aprendizaje supervisado en el mundo real.

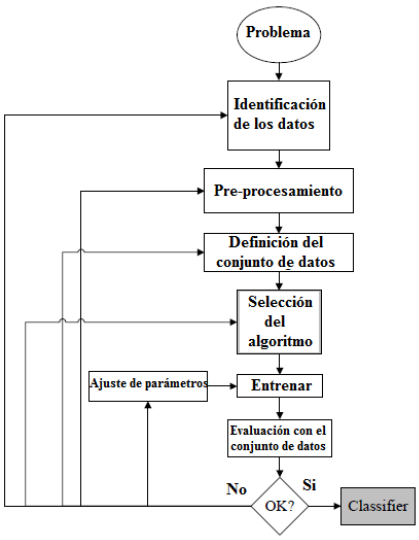


Figura 3.1.1.1 Diagrama de flujo aprendizaje supervisado

El primer paso es obtener los datos de entrenamiento, el cual, es un paso de gran importancia para obtener un clasificador funcional. Dentro del campo de la robótica, una técnica que ha sido muy utilizada para obtener el conjunto de datos de entrenamiento es el aprendizaje por demostración.

3.1.2 Aprendizaje por demostración

El aprendizaje por demostración es un subconjunto del aprendizaje supervisado [26]. Básicamente es un paradigma para transferir a los robots el conocimiento necesario para poder ejecutar una nueva tarea de forma autónoma [33]. La transferencia de conocimientos de un ser humano hacia un robot se puede realizar mediante tres tipos de ejemplos:

Ejemplos activos: Indican las demostraciones donde los usuarios realizan la tarea mientras que el robot rastrea el ambiente y realiza un mapeo entre el estado-acción. En este tipo de ejemplos se hace uso de herramientas tales como sensor de guantes, cámaras y sensores de tacto entre otros, lo cual, permite al robot hacer un sensado de la tarea. Entre algunos trabajos destacan [29] [30].

Ejemplos pasivos: Durante el proceso de demostración los sensores internos y externos del robot son seguidos y el sistema de aprendizaje intenta hacer un mapeo directo de las acciones a una acción bien conocida. El método requiere de menos inteligencia dado que la percepción está restringida a sensores bien conocidos y acciones disponibles del sistema. La principal desventaja radica en que la demostración está restringida al sistema objetivo.

Ejemplos implícitos: El usuario directamente especifica las metas que el sistema debe alcanzar. Esta es la metodología más sencilla ya que el mapeo con los símbolos está bien dado. Un beneficio es que el usuario está restringido a ciertas acciones y además se facilita si el robot puedes ser tele-operado tal como ocurre en [27].

En la presente tesis se hizo uso de ejemplos implícitos por las facilidades que ofrece la plataforma utilizada, así como la tarea objetivo que se desea alcanzar.

3.1.3 Generalización

El generalizar a través de un conjunto de datos inicial es el objetivo fundamental del aprendizaje de máquinas [31]. La generalización es la capacidad de un algoritmo de desempeñarse exitosamente en conjuntos de datos nuevos y desconocidos después de haber realizado un entrenamiento del mismo [32].

Para una generalización más exitosa se debe hacer una relación entre la complejidad de la hipótesis y la complejidad de los datos dados. Si la hipótesis es menos compleja que la función entonces tenemos sistema subentrenado, por ejemplo, si queremos realizar una clasificación de datos cúbicos mediante una línea recta. Por otra parte, si la hipótesis es más compleja que la función entonces tenemos sistema sobreentrenado, por ejemplo, si queremos realizar una clasificación de datos de una línea recta mediante un polinomio a la sexta [17].

Dentro de los algoritmos que permiten realizar una generalización destacan:

- Árboles de decisión
- Conjunto de reglas
- Redes neuronales artificiales
- Métodos bayesianos
- Máquinas de vector soporte
- K vecinos cercanos

Cada uno de los algoritmos mencionados anteriormente tiene diferentes ventajas y desventajas, las cuales, se detallan en la tabla 3.1.3.1, donde * representa mal desempeño y **** buen desempeño.

Tabla 3.1.3.1. Ventajas y desventajas de algoritmos de aprendizaje supervisado

	Árboles de decisión	Redes neuronales	Clasificador de Bayes	kNN	Máquina de vector soporte	Conjunto de reglas
Precisión en general	**	***	*	**	****	**
Velocidad de aprendizaje	***	*	****	****	*	**
Velocidad de clasificación	****	****	****	*	****	****

Tolerancia a valores faltantes	***	*	****	**	****	**
Tolerancia a valores irrelevantes	***	*	**	**	****	**
Tolerancia a atributos interdependientes	**	***	*	*	***	**
Tolerancia al ruido	**	**	***	*	**	*
Peligro de sobreentrenamiento	**	*	***	***	**	**
Habilidad de explicación	****	*	****	**	*	****
Manejo del parámetro de modelos	***	*	****	***	*	***

Generalmente las redes neuronales y las máquinas de vector soporte tienden a tener un mejor desempeño en problemas continuos y multidimensionales [26]. A pesar de que las máquinas de vector soporte muestran algunas ventajas sobre las redes neuronales artificiales de primera y segunda generación [36], en esta tesis se hizo uso de las redes neuronales pulsantes ya que superan en desempeño a las redes de primera y segunda generación y además ofrecen una buena cercanía a la realidad biológica.

3.1.4 Redes neuronales

Las redes neuronales son una abstracción de modelos inspirados en las células cerebrales y consisten en un conjunto de rasgos de entrada, por lo general binarios,

que son multiplicados por valores asociados a las neuronas llamados pesos sinápticos, por lo general reales entre 0 y 1, y finalmente insertadas a una función matemática que determina la activación de la neurona, tal como se muestra en la figura 3.1.4.1

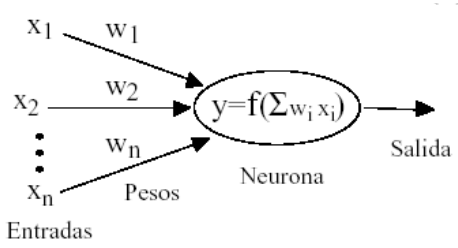


Figura 3.1.4.1. Neurona artificial

Al realizar el ajuste de los pesos sinápticos, mediante algoritmos de aprendizaje, se pueden obtener salidas deseadas ante entradas específicas. Existen diferentes algoritmos para ajustar los pesos y obtener el desempeño deseado de la red. Uno de los algoritmos más utilizados es el algoritmo de propagación hacia atrás, el cual, utiliza el método del gradiente descendiente.

El número de modelos de redes neuronales artificiales es muy alto. Desde el primer modelo propuesto por McCulloch y Pitts en 1943, se han desarrollado diferentes modelos con diferencias entre ellos tales como: funciones matemáticas de salida, valores aceptables de entrada, topologías, algoritmos de aprendizaje, entre otras. Todos estos modelos propuestos, recaen entre las primera y segunda generación de redes neuronales, las cuales, arrojan una salida binaria. Sin embargo, con el objetivo de poder resolver problemas de una forma más cercana a la forma de operar de un cerebro humano, actualmente, se está buscando migrar a modelos que se asemejen más al funcionamiento de las neuronas reales, tal como es el caso de las neuronas pulsantes.

3.1.5 Redes neuronales pulsantes

Las redes neuronales pulsantes recaen en la tercera generación de modelos de redes neuronales, incrementando el nivel de realismo en una simulación. Además de un

estado neuronal y sináptico, las redes neuronales pulsantes incorporan el concepto del tiempo al modelo. La idea es que las neuronas no se disparen en cada ciclo de propagación, tal como ocurre en las redes neuronales de perceptrón, en cambio, se disparan cuando un potencial de membrana alcanza un valor específico. Cuando una neurona se dispara genera una señal que atraviesa otras neuronas incrementando o decrementando su potencia de acuerdo a dicha señal.

Con el objetivo de mostrar una descripción detallada de las redes neuronales pulsantes utilizadas en la tesis, primeramente, se mostrará una introducción biológica sobre las neuronas pulsantes y más tarde el modelo matemático utilizado.

3.1.5.1 Neuronas pulsantes biológicas

Durante los últimos años, la investigación biológica ha acumulado un enorme conocimiento detallado acerca del funcionamiento y estructura del cerebro. El proceso elemental consiste en unidades llamadas neuronas, estas neuronas están conectadas entre sí en un patrón intrincado. En la realidad las neuronas corticales y sus conexiones están empacadas en una red densa con más de 10^4 cuerpos celulares y varios kilómetros de cables por milímetro cúbico [20].

Una neurona puede ser dividida en tres partes funcionales, llamada dendrita, soma y axón. Las dendritas juegan el papel de dispositivo de entrada, la cuales, colectan señales de otras neuronas y las transmiten al soma. El soma es la unidad central de procesamiento y lleva a cabo un paso no lineal de gran importancia: si el total de entradas exceden un cierto umbral, entonces la salida de la señal es generada. La señal de salida se transmite al dispositivo de salida llamado axón que le entrega la señal a otras neuronas.

La figura 3.1.5.1.1 muestra las principales partes de la neurona.

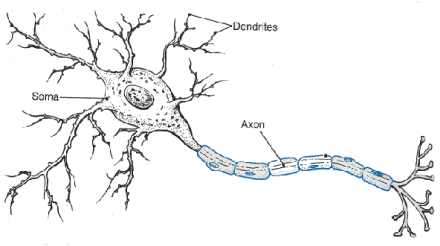


Figura 3.1.5.1.1. Partes de la neurona.

La unión entre dos neuronas se llama sinapsis. La neurona que envía la señal es llamada pre-sináptica y la receptora es llamada post-sináptica. Cada neurona comúnmente se conecta con más de 10^4 neuronas post-sinápticas. Algunas de estas ramas terminan en la neurona vecina, sin embargo, el axón se puede alargar varios centímetros para alcanzar otras neuronas en áreas más lejanas del cerebro.

Las señales de las neuronas consisten en pulso eléctricos cortos que pueden ser observados al poner un electrodo cerca del soma o axón de una neurona. Estos pulsos, llamados potencial de acción o pulsos, tienen una amplitud de 100 mV y típicamente una duración de 1-2 ms. Una cadena de potenciales de acción emitidas por una sola neurona es llamada tren de pulsos, lo cual es, una secuencia de eventos estereotipados que ocurren en un intervalo regular o irregular. Dado que todos los pulsos de una neurona dada son parecidos, la forma del potencial no acarrea ningún tipo de información. En cambio, es el número y tiempo de pulsos lo que importa. El potencial de acción es la unidad elemental de la señal de transmisión.

Las acciones de potencial de un tren de pulsos usualmente están bien separadas. Aunque exista una entrada muy fuerte, es imposible excitar un segundo pulso inmediatamente después del primero. La distancia mínima entre dos pulsos define el periodo de recuperación absoluto de la neurona. Dicho periodo esta seguido de una etapa donde es difícil, más no imposible, excitar el potencial de acción [4].

3.1.5.2 Tipos de neuronas pulsantes

Dentro de la corteza cerebral existen varios tipos de neuronas, las cuales son clasificadas de acuerdo al tipo de dinámica que muestran.

La figura 3.1.5.2.1 muestra diferentes tipos de neuronas pulsantes que se han estudiado en modelos matemáticos.

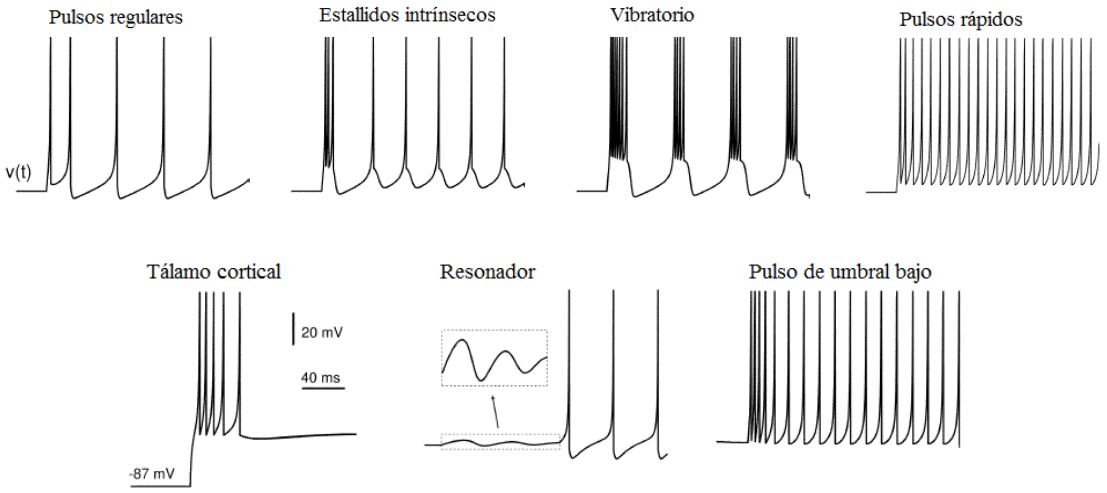


Figura 3.1.5.2.1. Tipos de neuronas pulsantes.

A continuación se describe la dinámica de cada tipo de neurona.

Pulsos regulares: Son las más encontradas en la corte cerebral. Cuando se les presenta un estímulo prologado las neuronas disparan un tren de 5 picos con un periodo corto entre los picos y después el período se incrementa. A esto se llama adaptación de la frecuencia de pulso.

Estallidos intrínsecos: Las neuronas disparan 5 pulsos continuos antes de disparar pulsos simples repetitivos.

Vibratorio: Las neuronas disparan pulsos muy cercanos. La frecuencia puede ser tan alta como 40Hz.

Pulsos rápidos: Las neuronas pueden disparar trenes periódicos de potencial de acción con extremadamente alta frecuencia prácticamente sin ninguna adaptación.

Pulsos de umbral bajo: Las neuronas pueden también generar trenes de pulsos de alta frecuencia de potencial de acción, pero con una pequeña frecuencia de adaptación. Estas neuronas tienen bajos umbrales de disparo.

Tálamo-cortical: Las neuronas tienen 2 regímenes de disparo. Cuando se resta (el voltaje es alrededor de -60mV) y después cuando se despolariza exhiben un disparo tónico.

Resonador: Las neuronas amortiguan o sostienen oscilaciones de sub-umbral. Las neuronas resuenan a las entradas rítmicas teniendo una frecuencia apropiada. Es importante notar que la neurona puede cambiar entre estados por un breve estimulo temporizado apropiado [3].

3.1.5.3 Modelo de Izhikevich

En años recientes han surgido modelos de neuronas pulsantes, los cuales, capturan la naturaleza de las neuronas y pueden producir varios patrones de pulsos [2]. Entre algunos modelos destacan Hodgkin-Huxley [19], Integrate and fire [34], FitzHugh-Nagumo [35], entre otros.

El modelo de Hodgkin-Huxley es uno de los más detallados y mejor conocidos de neuronas pulsantes. Describe de forma cuantitativa los comportamientos del nivel iónico subcelular y la corriente de membrana controlando la generación y propagación de los impulsos neurales. Aunque es biológicamente significativo y poderoso, el modelo es computacionalmente complejo para una simulación de una red neuronal a gran escala. Solamente modelos computacionalmente simples son prácticos a gran escala por su consumo limitado de hardware. Es por ello, que algunos modelos simplificados como *Integrate and fire* han sido propuestos como consecuencia. Sin embargo, a pesar de que estos modelos disminuyen significativamente el costo computacional, se despegan enormemente de la realidad [2].

La figura 3.1.5.3.1 detalla una comparativa entre los modelos de neuronas pulsantes más utilizados

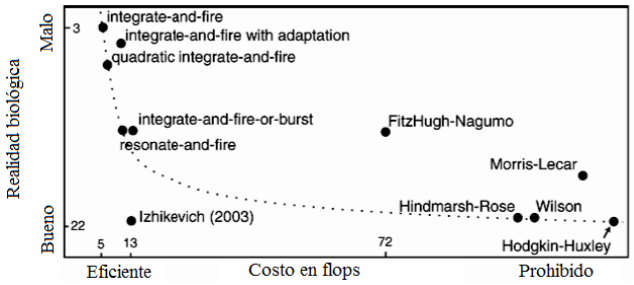


Figura 3.1.5.3.1. Comparativa entre modelos de neuronas pulsantes [35]

Tal como se puede apreciar en la figura 3.1.5.3.1, el modelo realizado por Izhikvevich tiene ventaja sobre los otros modelos propuestos ya que es un modelo simplificado que reduce el costo en flops y su aproximación a la realidad biológica es muy buena.

El modelo está basado en un par de ecuaciones diferenciales acopladas:

$$\begin{aligned} C\dot{v} &= k(v - v_r)(v - v_t) - u + I \\ \dot{u} &= a\{b(v - v_r) - u\} \end{aligned} \qquad \begin{aligned} &if \\ &v \geq v_{peak} \\ &then \\ &v \leftarrow c \\ &u \leftarrow u + d \end{aligned}$$

El modelo tiene 9 parámetros adimensionales. Dependiendo de los valores de a y b, puede ser un integrador o resonador. Los parámetros c y d toman en cuenta la acción de las corrientes del voltaje de umbral alto durante el pulso, y afectan el comportamiento post-pulso, v es el potencial de membrana, u es la corriente de recuperación, C es la capacitancia de la membrana, v_r es el potencial de membrana de descanso y v_t es el potencial de umbral instantáneo.

El signo de b determina si u es amplificador cuando (b<0) o resonador (b>0). La constante de tiempo de recuperación es a. El valor de corte del pulso es v_{peak}, y el voltaje de reset es c. El parámetro d describe el monto total de corrientes salientes menos las corrientes de entrada activadas durante el pulso.

Las dinámicas de otros tipos neuronales, incluyendo las del tronco cerebral, hipocampo, ganglios basales, y el bulbo también pueden ser descritas por el modelo [3].

Es importante mencionar que en este modelo la variable I representa una inyección de corriente a la neurona y, por lo tanto, se considera la variable de entrada en el modelo. Por otro lado, la variable v, la cual, representa el potencial de membrana de la neurona pulsante, es la salida del modelo tal como se puede apreciar en la figura 3.1.5.3.2.

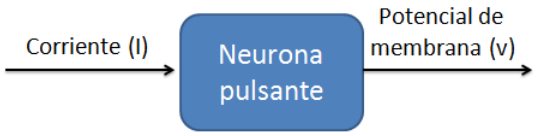


Figura 3.1.5.3.2. Entrada-salida del modelo de Izhikevich.

Como se mencionó en la sección 3.1.5, la neurona pulsante no genera una salida binaria, al contrario, genera como salida un tren de pulsos tal como ocurre en las neuronas reales. El tren de pulsos es originado por el potencial de membrana v.

Una vez que se inyecta una corriente I a la neurona pulsante, el potencial de membrana v empieza a cambiar su valor en función de los parámetros del modelo y de la corriente de recuperación u, del tal forma, que cuando v alcanza el valor de corte del pulso $v_{peak,}$ se considera que la neurona ha generado un pulso y el valor del potencial de membrana y de la corriente de recuperación reinicializan su valor.

3.1.5.4 Pulsos rápidos.

En el desarrollo de la presente tesis, se ajustaron los parámetros del modelo de Izhikevich para generar el patrón de disparo exhibido por los pulsos rápidos. La razón por la cual se realizó esta selección, fue que el tiempo transcurrido entre pulso y pulso es siempre el mismo, y por ello, el cálculo de la frecuencia en cualquier ventana de tiempo es directo. La figura 3.1.5.4.1 muestra el patrón exhibido por los pulsos rápidos.

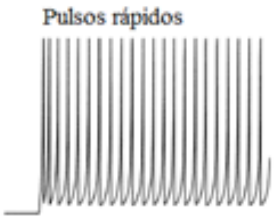


Figura 3.1.5.4.1. Pulsos rápidos.

El valor de los parámetros seleccionados se muestra a continuación:

$$\begin{aligned} 100\dot{v} &= 0.7(v-60)(v-40)-u+I \\ \dot{u} &= 0.03\{-2(v-60)-u\} \end{aligned} \qquad \begin{aligned} &if \\ &v \geq 35 \\ &then \\ &v \leftarrow -50 \\ &u \leftarrow u+100 \end{aligned}$$

Una vez que se inyecta la corriente I a la neurona pulsante, se genera un tren de pulsos mediante el potencial de membrana v, sin embargo, el presente modelo no cuenta con ningún parámetro que contabilice el número de pulsos generados; es por ello que fue necesario agregar un parámetro extra llamado nP, para almacenar el número de pulsos generados en un lapso de 1000 ms. El modelo final utilizado para la presente aplicación es el detallado a continuación:

$$\begin{aligned} 100\dot{v} &= 0.7(v-60)(v-40)-u+I \\ \dot{u} &= 0.03\{-2(v-60)-u\} \end{aligned} \qquad \begin{aligned} &if \\ &v \geq 35 \\ &then \\ &v \leftarrow -50 \\ &u \leftarrow u+100 \\ &nP \leftarrow nP+1 \end{aligned}$$

La tabla 3.1.5.4.1 muestra un ejemplo numérico de cómo se genera un pulso mediante el modelo descrito con un valor de corriente de entrada aleatorio.

Tabla 3.1.5.4.1. Ejemplo numérico de la generación de un pulso.

Ms	**Corriente de entrada *I***	**Potencial de membrana *v***	***dv/dt***	**Corriente de recuperación *u***	***du/dt***	***v >= 35***	**Número de pulsos generados**
1	642	-60	-53.57	0	0	No	0
2		-53.57	-47.75	0	-.385	No	
3		-47.75	-41.99	-.3855	-1.10	No	
4		-41.99	-35.80	-1.108	-2.15	No	
5		-35.80	-28.65	-2.1555	-3.54	No	
6		-28.65	-19.69	-3.542	-5.31	No	
7		-19.69	-7.49	-5.317	-7.57	No	
8		-7.49	10.95	-7.575	-10.4	No	
9		10.95	35	-10.49	85.55	Si	1
10		-50	-45.12	85.55	82.39	No	

Durante la ejecución del ejemplo anterior, el potencial de membrana v alcanzó el valor de corte en 91 ocasiones dentro de una ventana de tiempo de 1000 ms y por lo tanto se generaron 91 pulsos, los cuales, fueron almacenados en el parámetro nP.

La figura 3.1.5.4.1 detalla la evolución del ejemplo anterior dentro de la ventana de tiempo de 1000 ms. Es importante mencionar que todos los pulsos tienen exactamente la misma amplitud de acuerdo a la definición del modelo.

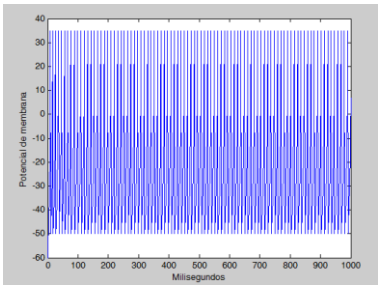


Figura 3.1.5.4.1. Ejemplo del tren de pulsos generado por la neurona de Izhikevich.

3.1.5.5 Cálculo de la corriente de entrada y frecuencia de salida.

La presente tesis hace uso de un vector de rasgos X y un vector de pesos W para obtener la corriente I y así calcular diferentes frecuencias para diferentes vectores de rasgos de entrada tal como se puede apreciar en la figura 3.1.5.5.1.

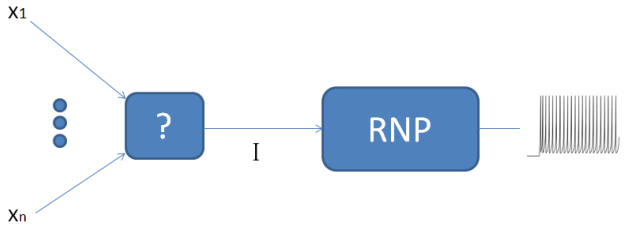


Figura 3.1.5.5.1. Conversión del vector de rasgos en corriente.

La corriente se calculó mediante el producto punto entre el vector de rasgos y los pesos sinápticos tal como se muestra a continuación:

$$I = \theta + x \cdot w$$

Existen un número infinito de operaciones que se pueden realizar para determinar la corriente de entrada, sin embargo, se ocupó el producto punto debido a que los rasgos utilizados en la presente tesis son linealmente separables.

Sin embargo, es importante resaltar que la operación que se aplica entre ambos vectores define la capacidad de la neurona, por ejemplo, si se hace uso del producto punto, entonces los problemas que la neurona será capaz de resolver deberán ser problemas linealmente separables, mientras que si se hace uso de una operación exponencial o logarítmica entonces se podrán resolver problemas no lineales.

Los valores de los parámetros que exhiben los pulsos rápidos en el modelo de Izhikevich, requieren una inyección de corriente I de 55pA para que la neurona pulsante se active. Para la presente aplicación, es fundamental que la neurona se active en todos los casos, es por ello, que se estableció un parámetro fijo, llamado θ, con valor de 55.

La figura 3.1.5.5.2 muestra un ejemplo numérico de cómo se transforma un vector de rasgos de 2 dimensiones a una corriente, por medio del uso del producto punto

entre el vector de rasgos y el vector de pesos. Una vez calculada la corriente, se introduce al modelo de Izhikevich y se obtiene la frecuencia final.

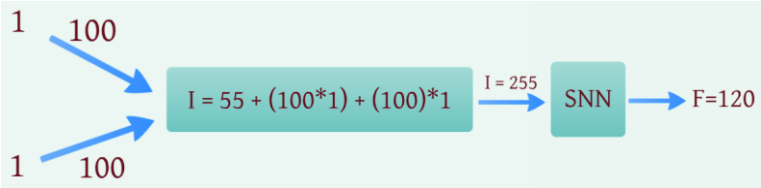


Figura 3.1.5.5.2. Ejemplo del cálculo de frecuencias con base en un vector de rasgos.

3.1.5.6. Clasificación mediante una neurona pulsante

El cálculo de una frecuencia en base a un vector de rasgos permite poder resolver problemas de clasificación. La presente aplicación, busca que vectores de rasgos pertenecientes a la misma clase generen aproximadamente la misma frecuencia, mientas que vectores de rasgos pertenecientes a diferentes clases deben generar una frecuencia lo suficientemente diferente como para poder discriminar entre las clases.

Lo anterior se reduce a uno de los problemas clásicos de las redes neuronales artificiales, el cual, consiste en realizar el ajuste de los pesos sinápticos, con base en un conjunto de datos de entrenamiento, hasta encontrar un hiperplano que permita realizar una clasificación exitosa para nuevos datos de entrada. Para el caso de la neuronales pulsantes, es importante poder encontrar un hiperplano que maximice la separación entre los vectores del conjunto de entrenamiento, de tal forma que cuando se presenten nuevos datos de entrada desconocidos, la clasificación pueda ser lo más precisa posible tal como se muestra en la figura 3.1.5.6.1.

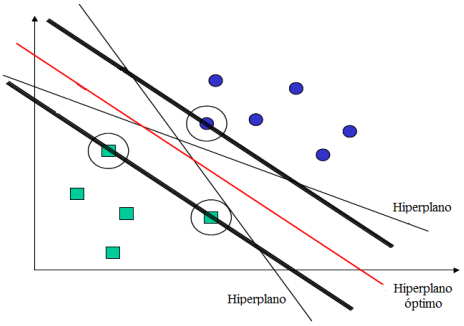


Figura 3.1.5.6.1. Hiperplano que maximiza la separación entre las clases.

A continuación se describe un ejemplo numérico de un problema de clasificación en dos dimensiones y su solución mediante una neurona pulsante.

La Tabla 3.1.5.6.1 muestra un ejemplo numérico de un conjunto de entrenamiento.

Tabla 3.1.5.6.1 Conjunto de entrenamiento para el ejemplo numérico de un problema de clasificación.

Rasgo 1	**Rasgo 2**	**Clase**
1	1	1
100	100	2

La Tabla 3.1.5.6.2 muestra un ejemplo numérico de un vector de pesos (calculado por un algoritmo explicado en la siguiente sección) que define un hiperplano en dos dimensiones y la frecuencia generada por cada vector del conjunto entrenamiento.

Tabla 3.1.5.6.2. Frecuencia generada por el conjunto de entrenamiento para el ejemplo numérico de un problema de clasificación.

Rasgo 1	**Rasgo 2**	**Peso 1**	**Peso 2**	**Frecuencia**
1	1	10.75	3.95	5
100	100	10.75	3.95	101

La Tabla 3.1.5.6.3 muestra un ejemplo numérico de una serie de vectores de prueba para observar cómo se resuelve el problema de clasificación mediante el uso de la distancia euclidiana mínima entre la nueva frecuencia generada y las frecuencias del conjunto de entrenamiento.

Tabla 3.1.5.6.3. Clasificación de los vectores de prueba para un ejemplo numérico.

Rasgo 1	**Rasgo 2**	**Peso 1**	**Peso 2**	**Frecuencia**	**Clase**
30	30	10.75	3.95	44	1
70	70	10.75	3.95	81	2
60	60	10.75	3.95	70	2
40	40	10.75	3.95	50	1

3.1.5.7. ¿Por qué usar redes neuronales pulsantes?

Diferentes algoritmos pudieron ser utilizados para resolver el problema abordado en la presente tesis, sin embargo, el uso de la neurona pulsante ofrece los siguientes beneficios:

- A pesar de que las computadoras actuales aventajan claramente en tiempo de procesamiento al cerebro humano, éste sigue siendo capaz de resolver problemas que los algoritmos existentes no pueden solucionar. Las neuronas pulsantes ofrecen una aproximación más cercana al comportamiento real de las neuronas de la corteza cerebral y una perspectiva más próxima de cómo el cerebro soluciona los problemas.

- Similar a una máquina con vectores soportes, una neurona pulsante permite mapear los datos a otro espacio dimensional. En el caso de la neurona pulsante, ésta los mapea al dominio frecuencial, donde, clases que pudiesen estar muy cercanas en el dominio original, se separan lo suficiente en el domino frecuencial para poder discriminar entre ellos. Esto se logra al ajustar los pesos sinápticos de la neurona mediante algoritmos exhaustivos, solución de ecuaciones diferenciales o cómputo evolutivo tal como lo realiza Roberto A. Vazquez [5].

- Una neurona pulsante permite que el sistema sea escalable dado que:

  - Ofrece un modelo adaptativo y, por lo tanto, el presente sistema puede ser llevado a un entrenamiento en línea donde se puedan agregar nuevos rasgos y nuevos valores en tiempo de ejecución. A diferencia de un árbol de decisión, por ejemplo, donde resulta más complicado agregar nuevos nodos.

  - A pesar de que una neurona de segunda generación es capaz de resolver el mismo problema, tal como se muestra en el apéndice D, un perceptrón, por definición, trabaja con el producto punto entre los rasgos de entrada y los pesos sinápticos. Por otra parte, una neurona pulsante no está restringida a trabajar con el producto punto y, por lo tanto, la operación de entrada llamada núcleo *(kernel)*, puede tomar diferentes formas y así dotar a una sola neurona pulsante de la capacidad de resolver problemas no lineales.

3.2 Visión por computadora

La visión por computadora consiste en obtener, procesar, analizar y entender imágenes del mundo real con el objetivo de producir información numérica o simbólica [38]. La visión por computadora está inspirada en duplicar las habilidades de la visión del ser humano [39]. Las aplicaciones de la visión por computadora se han dado desde tareas industriales hasta robots que comprenden su mundo circundante [40].

Un sistema de visión por computadora donde se desee realizar el reconocimiento de un objeto cuenta con los módulos mostrados en la figura 3.2.1 [41]:

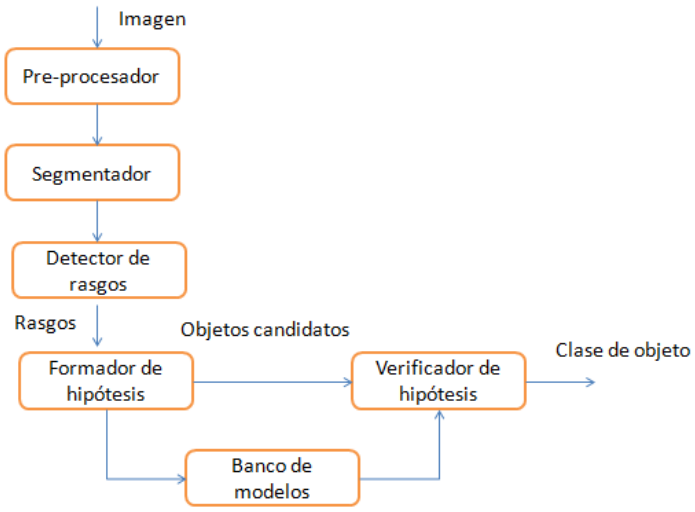


Figura 3.2.1. Componentes de un sistema reconocedor de objetos

Pre-procesado: Tiene como objetivo mejorar la calidad de la imagen para futuros tratamientos. El propósito es que etapas posteriores del análisis tengan mejores posibilidades de éxito.

Segmentador: Busca particionar o dividir la imagen en un conjunto de sub-imágenes de píxeles con la finalidad que cada sub-imagen aproxime en lo posible la región de cada uno de los objetos de la imagen.

Extractor de rasgos: Aplica operadores sobre una imagen permitiendo identificar rasgos que ayudan a formar una hipótesis sobre la presencia de un objeto en la escena.

Generador de hipótesis: Utiliza los rasgos extraídos de una imagen para asignar certidumbres a los objetos presentes en la escena correspondiente. Se puede utilizar métodos de cómputo suave durante esta etapa.

Módulo verificador: Hace uso de los modelos de los objetos contenidos en el banco de modelos para verificar las hipótesis generadas por el módulo generador de hipótesis.

Durante la presente tesis se hizo uso del sistema de visión HAVIMO 2.0, el cual, ya incorpora algunos de los módulos mencionados anteriormente, sin embargo, el presente sistema solamente está interesado en obtener los rasgos del objeto, por lo tanto, se prescindió del generador de hipótesis y del modulo verificador.

A continuación se hace una descripción de algoritmos y técnicas utilizadas por el sistema de visión HAVIMO 2.0.

3.2.1 Reconocimiento del color

Es la habilidad de una entidad de software o de hardware de poder distinguir objetos en base a su color, haciendo uso de la longitud de onda de luz que reflejan.

Por lo general se hace uso del espacio de colores RGB (rojo, verde, azul) con el objetivo de poder definir todos los colores como combinación de estos tres, sin embargo, tiene algunas desventajas tales como: alta correlación entre los canales, no-uniformidad perceptual significante y combinaciones de iluminación.

En la presente tesis se hizo uso del espacio de colores YC_bC_r, el cual, es un RGB no lineal codificado, donde el color ya es representado mediante la iluminación y construido como la suma de los pesos de los valores de RGB. La Y representa la iluminación del componente mientras que C_b y C_r son la diferencia de los componentes cromáticos de azul y rojo respectivamente. De esta forma se superan las barreras mencionadas anteriormente.

Una vez establecido el espacio de colores que se ha de utilizar es necesario aplicar los módulos vistos en la figura 3.2.1 para poder extraer los rasgos del objeto captado por la cámara.

3.2.2 Regiones crecientes

La segmentación es el proceso de dividir una imagen digital en varias partes con el objetivo de simplificar la representación de una imagen en otra más significativa y más fácil de analizar. La segmentación se usa para asignar una etiqueta a cada píxel de la imagen de forma que los píxeles que compartan la misma etiqueta también tendrán ciertas características visuales similares. Existen diferentes métodos para realizar la segmentación de una imagen, el sistema de visión HAVIMO 2.0 hace uso del método llamado regiones crecientes.

Regiones crecientes es un método de segmentación de imágenes basado en regiones simples, el cual, consiste en examinar los píxeles vecinos de los puntos iniciales llamados "puntos semilla" y determinar si dichos píxeles vecinos son parte de la región o no. Los puntos semilla, en este caso, están determinados por el color.

Algunas propiedades del algoritmo son las siguientes:

- La segmentación debe estar completa, es decir, todo píxel debe estar dentro de una región.
- Los puntos de una región deben estar conectados en un sentido predefinido, por ejemplo, en cuatro conexos u ocho conexos.
- Las regiones deben ser disjuntas.
- Una región solamente puede contener píxeles de la misma clase.
- Las regiones diferentes pertenecen a clases diferentes.

3.2.3. Extracción de rasgos

Una vez que se ha utilizado el método de segmentación por regiones crecientes, se procede a la etapa de extraer los rasgos. Los rasgos son un medio por el cual nosotros podemos definir objetos después de su segmentación.

Los rasgos utilizados para la presente tesis están basados en sus momentos geométricos cartesianos. Para el caso bidimensional f(x,y), el cual, supone ser una función continua por partes y con soporte compacto. El momento de orden $(p+q)$ se define como:

$$m_{pq} = \iint x^p y^p f(x,y) dx dy$$

En el caso de una imagen binaria con valores de {0,1} la ecuación toma la forma:

$$m_{pq} = \sum \sum x^p y^p$$

Dentro de la presente tesis se utilizaron rasgos derivados de estos momentos entre los cuales destacan:

Suma de píxeles en el eje x: Determina el tamaño de la base del objeto segmentado. Para obtener esta información se hace uso del momento *m10.*

Suma de píxeles en el eje y: Determina la altura del objeto segmentado. Para obtener esta información se hace uso del momento *m01.*

Área: Para imágenes de color, es la suma de todos los píxeles que tienen el mismo color correspondiente a la región. En el caso del área, se calcula mediante el momento *m00.*

Centroide: Determina exactamente el píxel que se encuentra en el centro de la imagen, es por ello, que es necesario tener un centro para el eje x y un centro para el eje y. Los centros son calculados de la siguiente manera:

$$cx = \frac{m_{10}}{m_{00}}$$

$$cy = \frac{m_{01}}{m_{00}}$$

3.3 Lógica difusa

La teoría de conjuntos fue introducida a principios de la década de los sesenta, por el Dr. Lofti A. Zadeh, profesor del Departamento de Ingeniería Eléctrica y Ciencias de la Computación perteneciente a la Universidad de California en Berkeley. La idea nace de la necesidad de dar solución a problemas de clasificación de categorías imprecisas, ya que en análisis de sistemas, las técnicas convencionales eran demasiado precisas o poco flexibles ante problemas del mundo real.

Durante las décadas de 1960 y 1970, Zadeh publicó artículos con toda la teoría que fundamenta la lógica difusa. En dichos artículos publicó los conceptos tales como funciones de membresía, variable lingüística, gráfica difusa, reglas difusas.

Dos o tres años más tarde Mamdani y Assilian publicaron un artículo, en el que demuestran el alcance de la teoría de lógica difusa como parte de un controlador, implementado en un control para un motor de vapor, con el uso de conjuntos difusos, variables lingüísticas y reglas difusas del tipo "Si (condición) entonces (realiza)", las cuales desde entonces se usan con éxito para dar solución a sistemas de control del mundo real.

3.3.1. Conjuntos difusos

Es un conjunto que puede contener elementos con grados parciales de membresía es decir, que sus elementos no le pertenecen de una manera absoluta, sino de forma graduada. El nivel de membresía de cada elemento a un conjunto difuso, indica lo que se denomina función de membresía, a diferencia de los conjuntos clásicos en donde un elemento pertenece o no pertenece, pues los métodos que usa la lógica matemática clásica son muy restrictivos.

3.3.2 Variable lingüística

Una variable lingüística, es una variable cuyos valores son palabras o sentencias en un lenguaje natural. Por ejemplo, la distancia que marca un espacio entre dos puntos, Distancia es una variable lingüística y sus valores son, "muy grande", "grande", "pequeña", "muy pequeña", "nula", es decir que se escogerán palabras de preferencia cortas que sean descriptivas con relación al significado de la variable lingüística elegida.

3.3.3. Funciones de membresía

Esta función está en el intervalo 0 y 1, es decir, esta función da para cada punto en el eje x (universo de discurso), un grado de membresía, siendo 1 el valor de membresía máximo. A continuación se enlistan las funciones de membresía utilizadas en la presente tesis.

Función triangular: Transforma un valor numérico x* en un conjunto difuso. La función de membresía está dada por:

$$\mu_A(A) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a < x \leq b \\ \frac{c-x}{c-b} & b < x \leq c \\ 0 & c < x \end{cases}$$

La figura 3.3.3.1 detalla la función de membresía triangular

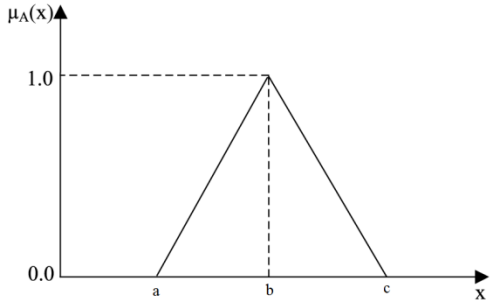


Figura 3.3.3.1 Función de membresía triangular

Función trapezoidal: Transforma un valor numérico real **x*** en un conjunto difuso **A**, el cual, posee la siguiente función de membresía trapezoidal:

$$\mu_A(A) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a < x \leq b \\ 1 & b < x \leq c \\ \frac{d-x}{d-c} & c < x \leq d \\ 0 & d < x \end{cases}$$

La figura 3.3.3.2 detalla la función de membresía trapezoidal.

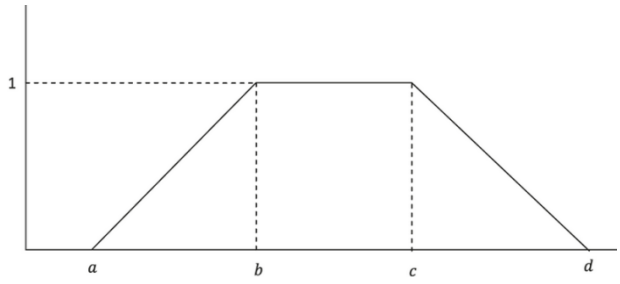


Figura 3.3.2.2. Función de membresía trapezoidal.

3.3.4. Base de reglas difusas

La base de reglas es la forma en que el sistema difuso almacena el conocimiento lingüístico que le permiten resolver el problema para el cual ha sido diseñado. La estructura de estas reglas es del tipo SI-ENTONCES, se muestra a continuación:

SI (la entrada es pequeña) ENTONCES (la salida es grande)
Antecedente Consecuente

El modelo formal de las reglas difusas esta dado por la siguiente expresión:

SI a_1 es A_1 y. . . y x_n es A_n ENTONCES $z=B_1$

En donde:

- a_1 es el antecedente, o variables lingüísticas de entrada.
- A_1. . . A_n y B_1 son conjuntos difusos, siendo R una relación discreta que puede definirse por una función típica que asigna 1 a cada combinación de elementos de universo que pertenecen a la relación y 0 a los que no pertenecen.
- $z\varepsilon V$ son las variables lingüísticas de salida.
- U es el conjunto universal de entrada.
- V es el conjunto universal de salida.

En una máquina de inferencia difusa los principios de lógica difusa se utilizan para combinar las reglas difusas SI-ENTONCES de la base de reglas difusas en una relación de un conjunto difuso A en U a un conjunto B en V.

Una regla difusa SI-ENTONCES es interpretada como una relación difusa en el espacio entrada-salida UxV.

3.3.5. Modelo de inferencia de Mamdani

El modelo de inferencia de Mamdani o también conocido como método de inferencia min-max, es el utilizado en la presente tesis como función de implicación.

La figura 3.3.5.1 muestra los componentes del modelo de inferencia tipo Mamdami.

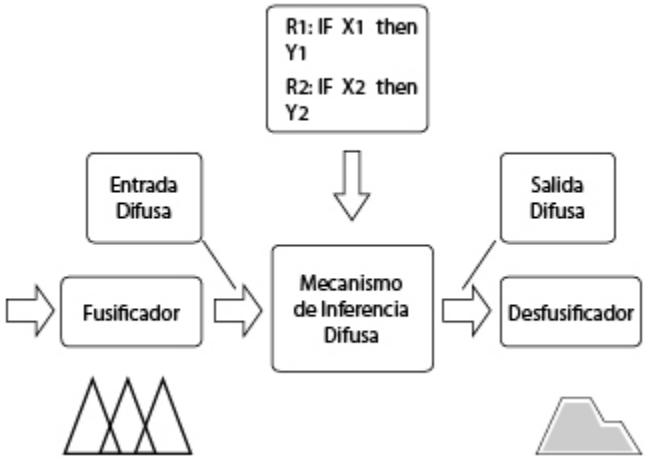


Figura 3.3.5.1. Modelo de inferencia tipo Mamdami.

En el mecanismo de inferencia los conjuntos antecedentes se someten a una operación min, mientras que los consecuentes obtenidos se someten a una operación max y se obtiene un conjunto difuso al cual se le aplica un defusificador para obtener un valor real.

3.4 BIOLOID Premium Kit

Dentro de la línea de modelos de la marca ROBOTIS se encuentra el modelo BIOLOID. Este modelo cuenta con diferentes Kits para su distribución y, a su vez, cada uno de ellos tiene diferentes características. La tabla 3.4.1 resalta las principales características de los Kits del modelo BIOLOID.

Tabla 3.4.1. Comparativa entre los KITS de BIOLOID

Descripción	**Bioloid Beginner Kit**	**Bioloid Comprehensive Kit**	**Bioloid Premium Kit**
Imagen			
Controlador	CM-5	CM-5	CM-510
DYNMYXL (servomotor)	AX-12A : 4EA	AX-12A : 18EA	AX-12A : 18EA
Sensor	AX-S1	AX-S1	GYRO, DMS, IR
S/W	SOFTWARE BIOLOID	SOFTWARE BIOLOID	RoboPlus
Alimentación	Pila NIMH 9.6V SMPS 12V 5ª	Pila 9.6V SMPS 12V 5A	Pila LIPO 11.1V SMPS 12V 5ª
Control remoto	RC-100 (Opcional) ZIG-100 set (Opcional)	RC-100 (Opcional) ZIG-100 set (Opcional)	RC-100 ZIG-110 set (Opcional)
Manual	Inicio Rápido (14 robots) Guía de usuario	Inicio Rápido (26 robots) Guía de usuario	Inicio Rápido (3 robots) e-manual (26 robots)

Para el presente trabajo se hizo uso del BIOLOID PREMIUM KIT debido a las ventajas que ofrece en comparación de los otros modelos entre las cuales resaltan:

- Excelente al caminar. Autoajusta su posición mientras camina.
- Permite programar el robot en lenguaje C o con ayuda del software especializado llamado Roboplus.
- Incluye sensores tales como infrarrojo, giro y distancia.
- Es posible incorporar dispositivos periféricos tales como la cámara para incrementar las capacidades del robot.

- Permite establecer una comunicación inalámbrica entre el robot y la computadora por medio de la tecnología Zigbee

El BIOLOID Premium Kit tiene la opción de armar humanoides de 3 tipos diferentes, cada uno de ellos con distintas opciones al caminar.

La figura 3.4.1 muestra los 3 tipos de humanoides. A la izquierda se ubica el tipo C, al centro el tipo A y a la derecha el tipo B

Figura 3.4.1. Tipos de humanoide del BIOLOID Premium Kit

La tabla 3.4.2 muestra una comparativa entre cada tipo de humanoide

Tabla 3.4.2. Comparativa de las características de los humanoides

Tipo	Características
A	- 18 servomotores - 1.7 kg - 39.7 cm - Pasos estables y omnidireccionales
B	- 16 servomotores - 1.6 kg - 39.7 cm - Capacidad de cambiar dirección mientras anda - 2 Dynamixeles disponibles para otras funciones
	- 16 servomotores - 1.6 kg - 38.6 cm

C	- Capacidad de ir de lado mientras camina. - 2 Dynamixeles disponibles para otras funciones.

Los principales componentes del BIOLOID Premium Kit se muestran en la figura 3.4.2.

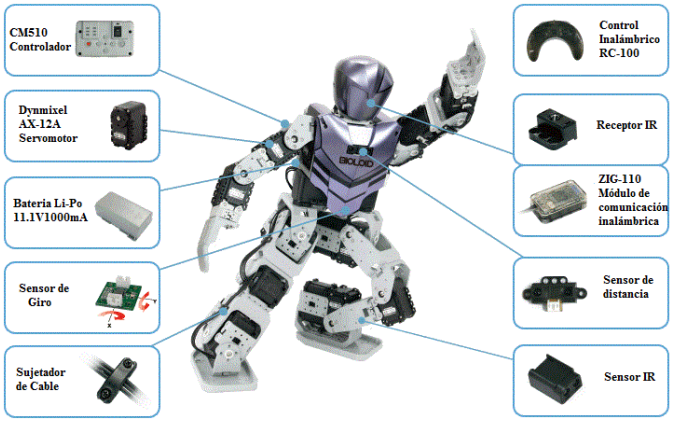


Figura 3.4.2. Principales componentes del BIOLOID Premium Kit

3.4.1 Dynamixel AX-12A

El BIOLOID PREMIUM KIT contiene un total de 18 dynamixeles AX-12A los cuales constituyen las articulaciones del robot humanoide y le permiten ejecutar movimientos.

La figura 3.4.1.1 muestra el dynamixel AX-12A

Figura 3.4.1.1. Dynamixel AX-12A

El dynamixel AX-12A es un servomotor digital que tiene la capacidad de controlar su velocidad, temperatura, posición, tensión y carga soportada, esto es posible

debido a que cada uno de ellos contiene un microcontrolador que entiende 50 comandos los cuales fijan y leen parámetros que definen el comportamiento del servomotor.

La tabla 3.4.1.1 muestra las principales características del dynamixel AX-12A

Tabla 3.4.1.1. Principales características del dynamixel AX-12A

Característica	**Valor**
Peso	54.5 gr.
Dimensiones	32mm*50mm*40mm
Resolución	29°C
Torque	15kgf.cm(con 12.0V y 1.5A)
Voltaje de alimentación	9 ~ 12V (Recomendado 11.1V)
Temperatura	-5°C ~ +70°C
ID	(0~253)
Velocidad de comunicación	7343bps ~ 1 Mbps
Material	Plástico de ingeniería

3.4.2 CM-510

La figura 3.4.2.1 muestra la vista frontal del CM510

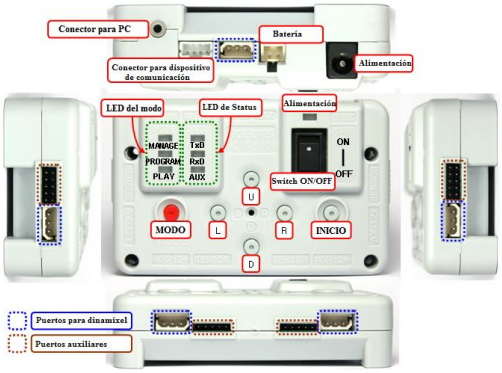


Figura 3.4.2.1. Controlador CM510

El CM510 es el cerebro del robot. Contiene el microcontrolador Atmega2561 y entre sus funciones destacan:

- Coordinar los movimientos del robot.
- Obtener el voltaje necesario para alimentar todos los componentes.
- Permitir programar diferentes movimientos y tareas.
- Fungir como interfaz entre la computadora y periféricos.

La tabla 3.4.2.1 muestra las principales características físicas y eléctricas del CM-510.

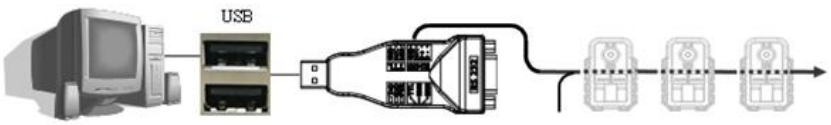
Tabla 3.4.2.1. Características del CM-510.

Característica	**Valor**
Peso	51.3 gr.
Microcontrolador	ATMega2561
Voltaje de alimentación	6.5V ~ 15V (Recomendado 11.1V)
Corriente de alimentación	50 Ma
Corriente máxima soportada	10A(Fusible)
Temperatura	-5°C ~ +70°C

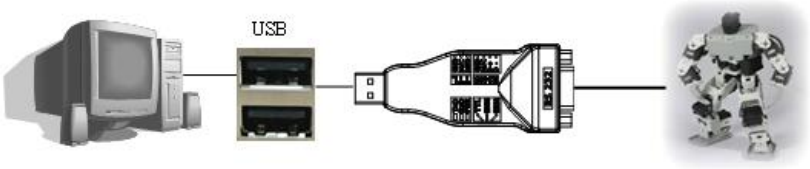
3.4.3 USB2Dynamixel

Dado que la comunicación entre el BIOLOID Premium Kit con la PC es de forma serial, y además, la mayoría de las computadoras actuales no tienen este tipo de puerto, el Kit contiene un componente llamado USB2Dynamixel, el cual, es un dispositivo que funge como interfaz y permite controlar cada uno de los dynamixeles a través de la PC por medio de su conexión al CM-510.

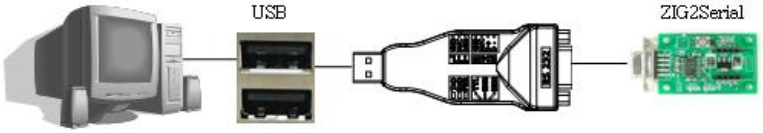
La figura 3.4.3.1 detalla el uso del USB2Dyamixel para realizar la comunicación entre el robot y la PC.



(a)



(b)



(c)

Figura 3.4.3.1. a) Control de dynamixeles usando PC b) Controlando por del puerto serial c) Comunicación inalámbrica.

3.4.4 Zig-100/110

La figura 3.4.4.1 muestra los módulos Zig-100/110.

Figura 3.4.4.1. Zig-100/110.

El Zig-100/110 son un par de módulos que permiten realizar la comunicación inalámbrica entre el robot y el control inalámbrico, durante la ejecución de tareas. La comunicación se logra mediante la tecnología Zigbee, la cual, permite enviar y recibir datos en un área geográfica pequeña.

Con el objetivo de realizar la comunicación inalámbrica entre la PC y el robot es necesario hacer uso de un módulo extra llamado Zig2serial el cual se muestra en la figura 3.4.4.2.

Figura 3.4.4.2. Zig2Serial.

A este módulo es necesario conectarle el Zig-100 tal como lo muestra la figura 3.4.4.3.

Figura 3.4.4.3. Conexión entre Zig2Serial y Zig-100.

Finalmente, con el objetivo de lograr la comunicación inalámbrica, se conecta el Zig2Serial al USB2Dynamixel y este a la computadora tal como se muestra en la figura 3.4.4.4.

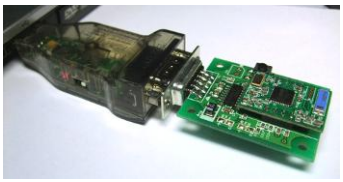


Figura 3.4.4.4. Conexión entre el Zig2Serial y el USB2Dynamixel.

3.4.5 Sensor de distancia

Es un sensor utilizado para detectar objetos o paredes en una cierta distancia. La figura 3.4.5.1 muestra el sensor de distancia.

Figura 3.4.5.1. Sensor de distancia.

La figura 3.4.5.2 muestra la relación que existe entre los valores arrojados por el software de roboplus y la distancia en centímetros de un objeto cercano al sensor.

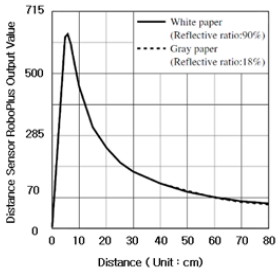


Figura 3.4.5.2. Distancia vs valor arrojado por el sensor de distancia.

3.4.6 Sensor de giro

Es el sensor que mide la velocidad angular, calcula en qué lado el robot está poniendo más potencia. Este sensor permite garantizar que el robot camine de forma estable y sin caerse constantemente. La figura 3.4.6.1 muestra el sensor de giro del BIOLOID Premium Kit.

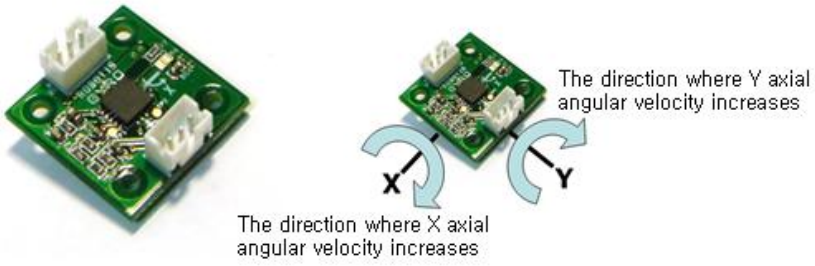


Figura 3.4.6.1. Sensor de giro.

3.4.7 Roboplus

Roboplus es un software especializado para el BIOLOID Premium Kit que permite manipular, administrar y programar al robot mediante una PC. Las tres aplicaciones que Roboplus contiene para poder realizar esto son:

Roboplus Task: Permite crear archivos tipo task, con los cuales, podemos programar bifurcaciones, ciclos, funciones, entre otros dentro de la memoria del robot.

La figura 3.4.7.1 muestra la aplicación del Roboplus Task.

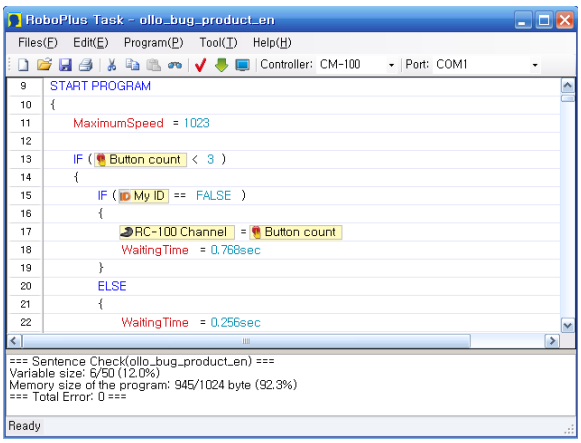


Figura 3.4.7.1. Roboplus Task.

Roboplus Motion: Permite crear archivos tipo motion, con los cuales, podemos programar los movimientos del robot controlando los actuadores. Con este tipo de archivos podemos programar tanto la posición como la velocidad de cada uno de los 18 actuadores.

La figura 3.4.7.2muestra la aplicación del Roboplus Motion.

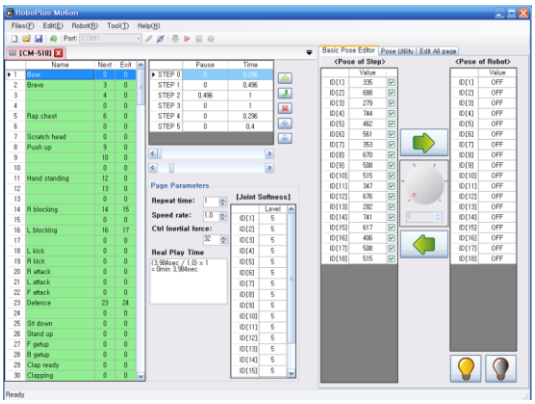


Figura 3.4.7.2. Roboplus Motion.

Roboplus Manager: Permite administrar cada componente del dispositivo. Mediante esta aplicación se pueden ajustar y modificar cada una de las propiedades de los dynamixeles, sensores y otros periféricos. Además se puede restaurar el firmware de los componentes y reinicializar sus valores de fábrica.

La figura 3.4.7.3 muestra la aplicación del Roboplus Manager

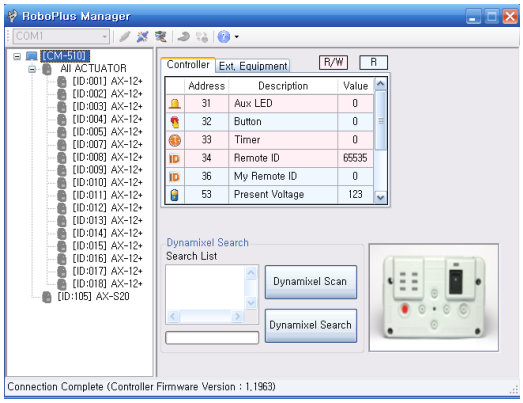


Figura 3.4.7.2. Roboplus Manager.

3.4.8 Cámara HAVIMO 2.0

La cámara HAVIMO 2.0 es un sistema de visión por computadora para microprocesadores de baja potencia. El sistema hace el procesamiento de la imagen y los resultados son obtenidos vía serial. La figura 3.4.8.1 muestra la cámara HAVIMO 2.0.

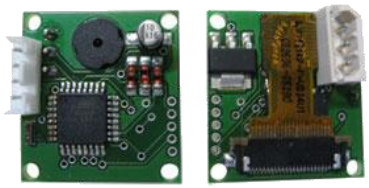


Figura 3.4.8.1. Cámara HAVIMO 2.0.

El algoritmo utilizado para realizar el análisis de imágenes es el de regiones crecientes. HAVIMO 2.0 tiene la gran ventaja que es un sistema que es compatible con ROBOTIS y puede ser integrado con los programas de RoboPlus.

La tabla 3.4.8.1 muestra las principales características de la cámara.

Tabla 3.4.8.1. Principales características de la cámara HAVIMO 2.0.

Característica	**HAVIMO 2.0**
Resolución	160*120 pixeles
Cuadros por segundo	19 Fps
Número de colores detectables	Hasta 256 colores
Regiones detectables por cuadro	Hasta 15 regiones
Tipo de comunicación	Serial
Software soportado	Roboplus (ROBOTIS) Programación en C Firmware RBC
Hardware soportado	CM-5 CM-510 USB2Dyanmixel

Con el objetivo de utilizar la cámara HAVIMO 2.0 ésta debe ser calibrada mediante el uso de una computadora. La figura 3.4.8.2 detalla las diferentes formas en las cuales se puede realizar la conexión entre la cámara HAVIMO y la computadora.

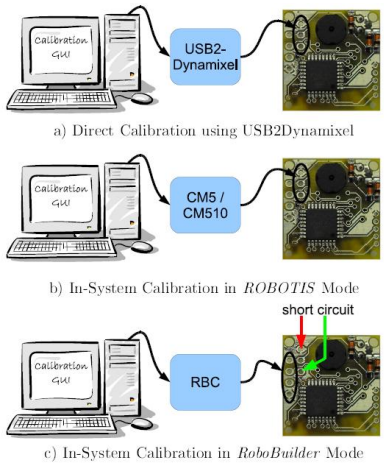


Figura 3.4.8.2. Conexión entre la cámara HAVIMO 2.0 y la computadora.

Con el objetivo de realizar la calibración de la cámara HAVIMO 2.0 con una PC es necesario descargar un software especializado llamado HAVIMO_GUI el cual puede ser adquirido en la siguiente página de Internet: *http://sourceforge.net/projects/havimo/.*

La figura 3.4.8.3 muestra la interfaz usada por HAVIMO_GUI.

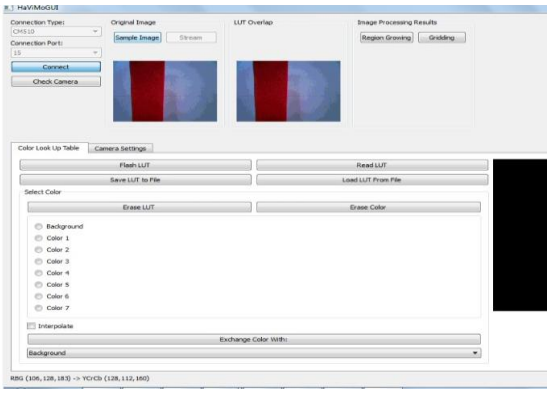


Figura 3.4.8.3. HAVIMO_GUI.

Dado que la cámara HAVIMO 2.0 es compatible con el robot humanoide, es posible acceder a la información procesada por la cámara durante la ejecución de un programa en RoboPlus Task. La figura 3.4.8.4 detalla la información a la que se puede acceder durante la ejecución de un programa.

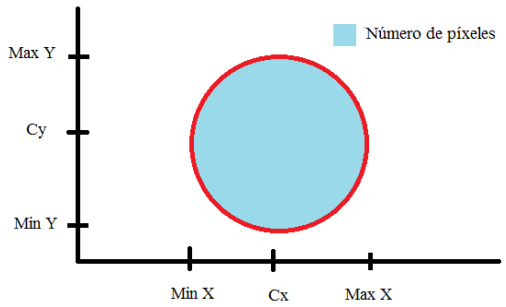


Figura 3.4.8.4. Datos accesibles desde un programa en RoboPlus Task.

A pesar de que la cámara HAVIMO 2.0 es compatible con el robot BIOLOID y mediante un programa se pueden acceder a los datos, no es posible acceder a la imagen capturada.

3.4.9 Montaje de la cámara HAVIMO 2.0 en el BIOLOID Premium Kit

A pesar del hecho que la cámara HAVIMO 2.0 es compatible con el BIOLOID Premium Kit, ésta no es parte del Kit original y por lo tanto no existe un lugar designado en la estructura del robot para colocarla.

Debido a ésta limitante, se realizaron algunas modificaciones en la estructura original del robot. El objetivo fue obtener un sistema más flexible que permitiera al robot humanoide mover la cámara HAVIMO 2.0 tanto en un plano transversal como coronal. La figura 3.4.9.1 muestras las modificaciones que se realizaron en la estructura del robot humanoide.

Figura 3.4.9.1. Modificaciones estructurales en el robot humanoide.

Tal como se puede apreciar en la figura 3.4.9.1, se removieron los dos motores que se encontraban en los antebrazos de la estructura original y se colocaron en la

cabeza del robot. Así mismo, la cámara HAVIMO 2.0 se montó en uno de los motores colocados en la cabeza.

De ésta forma, se creó un sistema más flexible, otorgando al robot la capacidad de hacer un barrido del mundo exterior mediante el movimiento de los motores montados en la cabeza y la continua captura de imágenes por parte de la cámara HAVIMO 2.0.

3.5 Java Speech

El API de Java Speech (JSAPI) es una extensión de la plataforma Java, la cual, define una interfaz de software multiplataforma estándar y fácil de usar para implementar la tecnología de habla.

Java Speech ofrece las siguientes ventajas:

- Provee soporte para sintetizadores de voz y para sistemas de reconocimiento de habla.
- Provee una interfaz multiplataforma y multiproveedor robusta para la síntesis y reconocimiento del habla.
- Permite el acceso al estado del arte de la tecnología del habla.
- Es simple, compacto y fácil de aprender.

La figura 3.5.1 muestra las partes principales del API.

Figura 3.5.1. Partes del API de JavaSpeech.

A pesar de que JavaSpeech puede ser utilizado para síntesis, la presente tesis solo hace uso del motor reconocedor de voz para poder transmitir el comando de forma verbal.

El API funciona mediante la definición de una gramática que debe seguir un formato específico. Primeramente se deben definir los datos y más tarde las posibles combinaciones que pueden existir entre los datos.

Un ejemplo del formato se detalla a continuación.

Datos:
<dato1>=X;
<dato2>=Y;
<dato3>=Z;

Combinaciones:
[<dato1><dato2>]
[<dato1><dato3><dato2>]
[<dato3>]

Gramática que el sistema reconocerá:

XY
XZY
Z

Capítulo 4

Metodología

En este capítulo se presenta a detalle la metodología desarrollada para realizar el aprendizaje supervisado que permita al robot humanoide BIOLOID desplazarse y manipular objetos. De igual forma se muestran los procedimientos y conceptos más relevantes bajo los que se construyeron los algoritmos que combinan las técnicas del área de visión por computadora, redes neuronales pulsantes y control difuso, para la ejecución de las acciones dirigidas a cumplir con los objetivos de esta tesis.

4.1 Descripción del sistema

El sistema cuenta con dos módulos: la computadora, que a su vez contiene la neurona pulsante, el sistema de control difuso y el API de Java Speech, los cuales, son utilizados para clasificar la acción del robot de acuerdo al patrón de entrada, calcular el tiempo que la acción será ejecutada y reconocer del comando verbal respectivamente. Por otra parte, el sistema cuenta con un robot humanoide BIOLOID que envía los rasgos capturados del medio hacia la computadora tal como se puede apreciar en la figura 4.1.1.

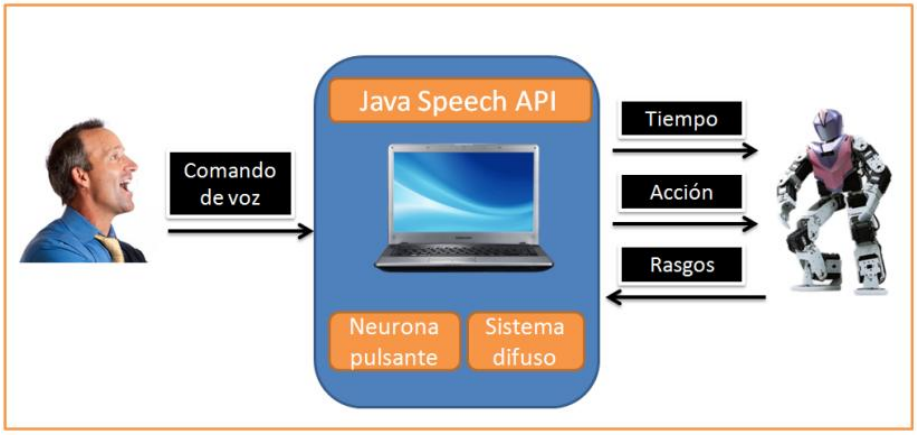


Figura 4.1.1. Módulos del sistema.

4.2 Arquitectura del sistema

La arquitectura del sistema esta detallada en la figura 4.2.1.

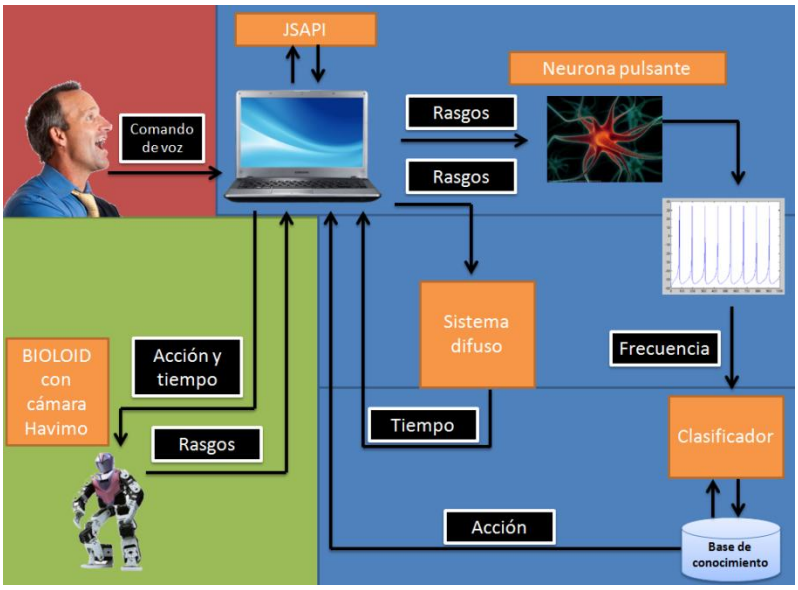


Figura 4.2.1. Arquitectura general del sistema.

Tal como se puede apreciar en la figura 4.2.1, él sistema contiene 2 módulos generales: el robot (verde) y la computadora (azul).

En un inicio, el robot humanoide se coloca en un punto aleatorio del espacio de trabajo al igual que los 2 objetos descritos en las condiciones iniciales del capítulo 1. Cuando el usuario pronuncia un comando verbal, la computadora hace uso del API de Java Speech para transformarlo a la instrucción que define cual será la tarea a realizar y se envía la orden de activación al robot.

El robot humanoide BIOLOID hace la búsqueda del objeto deseado mediante un movimiento en el plano transversal del motor que tiene montado el sistema de visión HAVIMO 2.0, el cual, toma fotos cada 0.2 segundos hasta localizar el objetivo. Una vez localizado el objeto, el robot toma la información capturada por la cámara y los sensores y la transmite de forma inalámbrica a la computadora mediante el módulo Zig 100. La información recibida por la computadora se introduce a la neurona pulsante y al sistema difuso con el objetivo de obtener la acción correspondiente y el tiempo que será ejecutada. Finalmente los datos son enviados al robot y el ciclo se repite hasta que se logra la tarea especificada por el comando verbal del usuario.

4.3 Módulo: Robot humanoide

El módulo del robot consiste en dos partes. Primeramente en la extracción de rasgos, los cuales, son la información capturada del entorno por medio de la cámara, el sensor de distancia y la posición de los motores. La segunda parte consiste en la acción a ejecutar y el tiempo de ejecución definidos por la computadora.

4.3.1 Rasgos

Con el objetivo de extraer información relevante acerca de la posición actual del robot con respecto al objeto a tomar o a la posición meta, fue necesario seleccionar rasgos de orientación, cercanía y posición.

La figura 4.3.1.1 muestra lo descrito anteriormente.

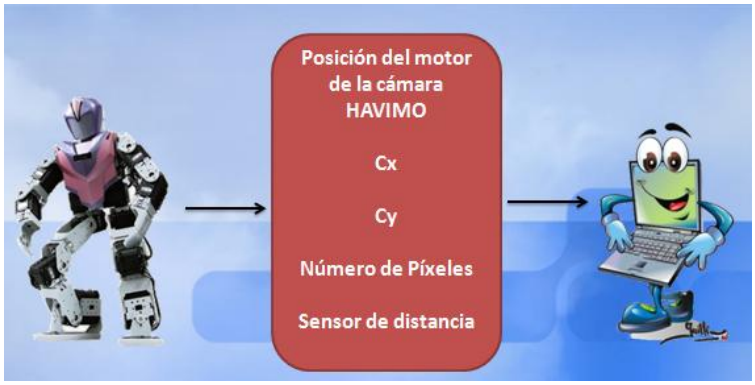


Figura 4.3.1.1. Rasgos seleccionados.

A continuación se describe en qué consiste cada uno de los rasgos.

Posición del motor con movimiento transversal: Cuando el sistema de visión HAVIMO 2.0 montado en un motor con movimiento transversal localiza el objeto, el motor, automáticamente se detiene y se guarda la posición actual del mismo. Este rasgo es de gran importancia ya que permite saber si el objeto esta hacia la izquierda o derecha del robot y además en qué medida.

Centroide x y y: Permite posicionar al robot con respecto al objeto para tormarlo. Este rasgo ofrece el detalle cuando el robot se encuentra más cercano al objeto.

Sensor de distancia: Este rasgo es el que le permite al robot conocer su distancia con respecto al objeto a tomar, en caso de encontrarse al menos a 30 cm del mismo.

Número de píxeles: Este es un rasgo que es utilizado una vez que el objeto ya ha sido tomado y se está buscando llevarlo hasta la posición meta. Dado que el sensor de distancia se inhabilita, la única manera que el sistema puede saber la cercanía con respecto a la meta está basada en el número de píxeles obtenidos por el sistema de visión HAVIMO 2.0.

Algunos rasgos recibidos por la computadora son introducidos a la red neuronal pulsante para obtener la acción a realizar y otros al sistema difuso para saber cuánto tiempo se ejecutará la acción.

La figura 4.3.1.2 detalla los rasgos introducidos a la red neuronal pulsante y los rasgos introducidos al sistema difuso.

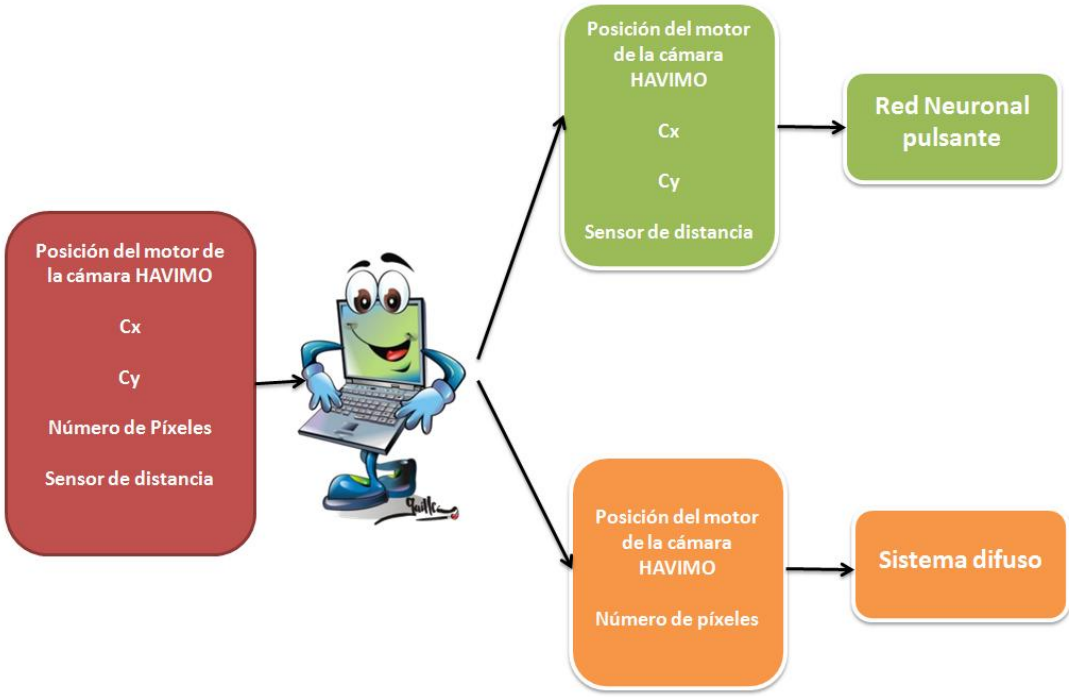


Figura 4.3.1.2. División de los rasgos obtenidos por la computadora.

4.3.2 Acciones del robot

Con el objetivo de poder ejecutar la tarea, el robot se debe desplazar con una orientación, una dirección y un sentido. Para lograr esto, es necesario programar valores en cada uno de los 18 actuadores que le permitan al robot poder ejecutar diferentes acciones mediante el software de RoboPlus Motion.

Las acciones definidas que el robot puede ejecutar son las mostradas en la figura 4.3.2.1.

Figura 4.3.2.1. Posibles acciones del robot humanoide.

Avanzar: El robot se desplaza hacia adelante en el plano frontal.

Girar a la derecha: El robot realiza un movimiento rotatorio sobre su propio eje hacia el lado derecho de aproximadamente 30°.

Girar a la izquierda: El robot realiza un movimiento rotatorio sobre su propio eje hacia el lado izquierda de aproximadamente 30°.

Lateral derecha: El robot realiza un desplazamiento hacia su costado derecho sin girar en su eje.

Lateral izquierda: El robot realiza un desplazamiento hacia su costado izquierdo sin girar en su eje.

Tomar objeto: El robot ejecuta la acción de agacharse, abrir los brazos y cerrarlos. De esta forma es posible realizar la toma del objeto.

4.4 Módulo: Computadora.

El módulo computadora cuenta con tres sub-módulos para procesar la información proveniente del robot y el usuario. La figura 4.4.1 muestra los sub-módulos de la computadora.

Figura 4.4.1 Sub-módulos de la computadora.

4.4.1 Neurona pulsante.

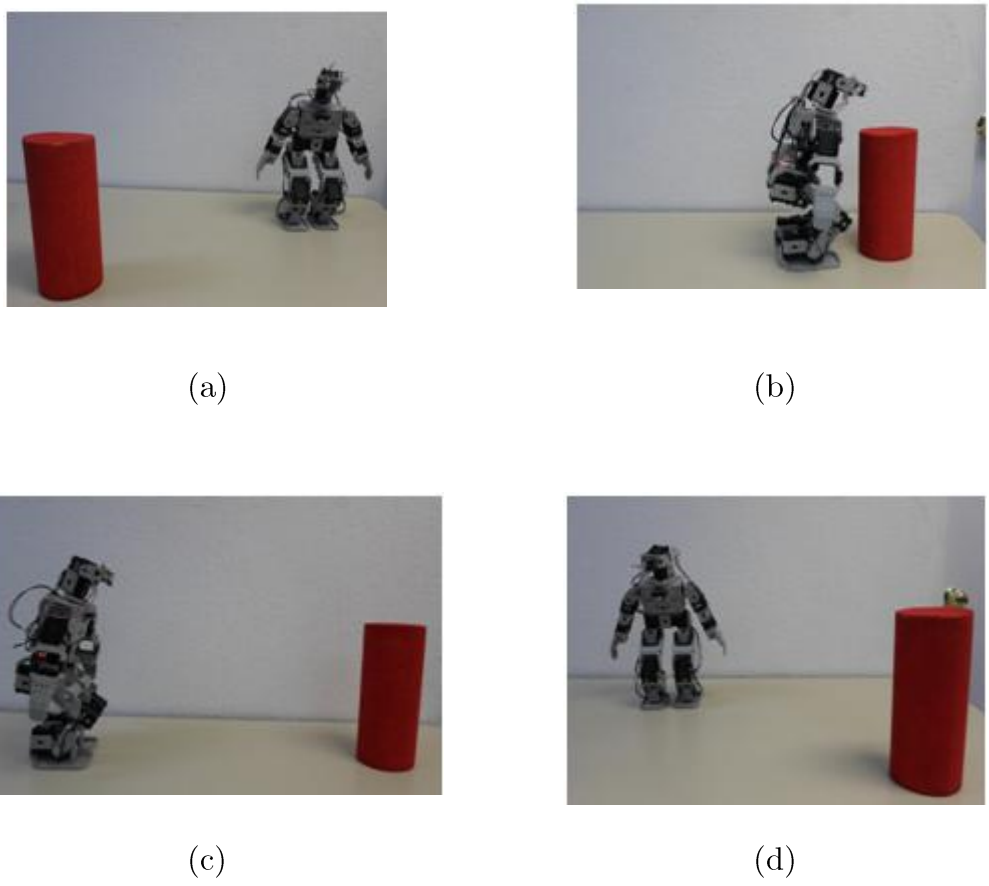
La neurona pulsante de Izhikevich es un sub-módulo de la computadora. Debido a su capacidad de generalización, es utilizada para clasificar los rasgos enviados por el robot y obtener la acción correspondiente a realizar.

La neurona pulsante de Izhikevich cuenta con un conjunto de entrenamiento y, mediante un algoritmo exhaustivo de ajuste de pesos sinápticos, calcula el hiperplano que permite realizar una clasificación exitosa para nuevos datos de entrada.

4.4.1.1 Conjunto de entrenamiento.

Con el propósito de crear el conjunto de entrenamiento, se posicionó al robot y al objeto en diferentes configuraciones, de tal forma, que el robot realizaba la búsqueda del objeto y, en el momento de localizarlo, los rasgos descritos en la sección anterior se almacenaban. De acuerdo a la percepción del usuario, dichos datos fueron etiquetados con una acción correspondiente.

La figura 4.4.1.1.1 muestra como se realizó la captura de los datos y las etiquetas que se otorgaron para dicha posición de acuerdo al punto de vista del usuario.



(a) (b)

(c) (d)

Figura 4.4.1.1.1. Construcción del conjunto de entrenamiento. a) Girar derecha. b) Tomar objeto. c) Avanzar. d) Girar izquierda.

Para facilitar la captura de los datos en la PC, se realizó una interfaz la cual se muestra en la figura 4.4.1.1.2.

Una vez capturados los datos, se calculó el promedio de los datos pertenecientes a cada clase. Finalmente la tabla 4.4.1.1.1 muestra el conjunto de entrenamiento etiquetado de acuerdo a la percepción del usuario.

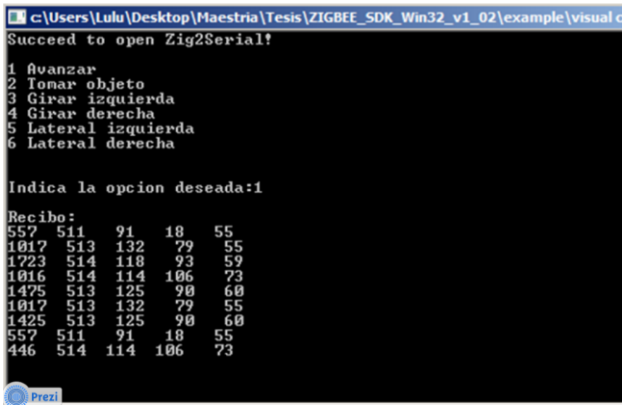


Figura 4.4.1.1.2 Interfaz para capturar datos

Tabla 4.4.1.1.1. Conjunto de entrenamiento de la red neuronal pulsante.

Acción	Posición Motor	Cx	Cy	Sensor de distancia
Avanzar	514	118	93	59
Girar derecha	416	141	19	61
Girar izquierda	591	16	17	65
Lateral derecha	255	133	87	66
Lateral izquierda	756	12	90	80
Tomar objeto	514	125	118	400

4.4.1.2 Entrenamiento de la neurona pulsante.

Con el objetivo de obtener una clasificación exitosa con una neurona pulsante de Izhikevich, es necesario ajustar los pesos sinápticos, de tal forma, que se calcule un hiperplano que separe las clases (acciones), lo suficiente para poder discriminar entre ellas.

Actualmente no existe un método generalizado para ajustar los pesos sinápticos de las redes neuronales pulsantes como en el caso de la propagación hacia atrás en las redes de perceptrón. Es por ello que en la presente tesis, se usó un método exhaustivo para realizar el ajuste de los pesos sinápticos.

El algoritmo explora todas las combinaciones posibles entre los elementos del vector de pesos, el cual, tiene tantos elementos como el vector de entrada.

El *peso inicial* y el *cambio de peso* son parámetros dinámicos y quedan disponibles para que el usuario los defina. En caso de que el usuario quiera alcanzar un mayor nivel de detalle y explorar más pesos entonces el *cambio de peso* deberá ser menor y viceversa.

La variable *times* indica la profundidad con la cual se desea explorar las posibles combinaciones en el vector de pesos.

Una vez que se obtiene una nueva configuración de pesos, se prueban todos los datos del conjunto de entrenamiento en la red neuronal pulsante, la cual, calcula el tren de pulsos generado por cada vector. Todas las frecuencias son almacenadas y se calcula la distancia, basada en el número de pulsos en una ventana de tiempo de 1000 ms, que existe entre ellas; el algoritmo devuelve la distancia más pequeña entre las clases en la iteración actual, ya que ésta indica que tan bueno es el hiperplano obtenido. Lo anterior se detalla en la tabla 4.4.1.2.1, donde se muestra un ejemplo, con 4 clases, la frecuencia generada y la distancia entre cada clase.

Tabla 4.4.1.2.1. Ejemplo numérico del cálculo de la distancia entre clases.

Clase	**Frecuencia**	**Distancia con clase 1**	**Distancia con clase 2**	**Distancia con clase 3**	**Distancia con clase 4**
1	5		2	6	15
2	7	2		2	13
3	11	6	4		9
4	20	15	13	9	

En este caso la menor distancia obtenida es 2 y será la distancia almacenada.

El proceso se repite, un nuevo vector de pesos es generado, y con él, un nuevo hiperplano. La distancia más pequeña entre las clases se calcula de nuevo y se compara con la última obtenida, si es mayor se guardan los pesos correspondientes y el ciclo se vuelve a iniciar.

El proceso se interrumpe en caso de que una las siguientes condiciones se cumpla:

- La distancia deseada por el usuario ya se alcanzó.
- Se alcanza la profundidad máxima, establecida por el usuario, en cada uno de los elementos del vector de pesos.
- Se alcanzan 20 mil épocas donde no se encuentra un mejor hiperplano.

Finalmente tenemos almacenados en el vector *Pesos Max* el valor de pesos sinápticos que genera la máxima separación lineal entre las clases obtenida por el algoritmo.

La figura 4.4.1.2.1 muestra la aplicación del algoritmo en el caso 3 dimensional.

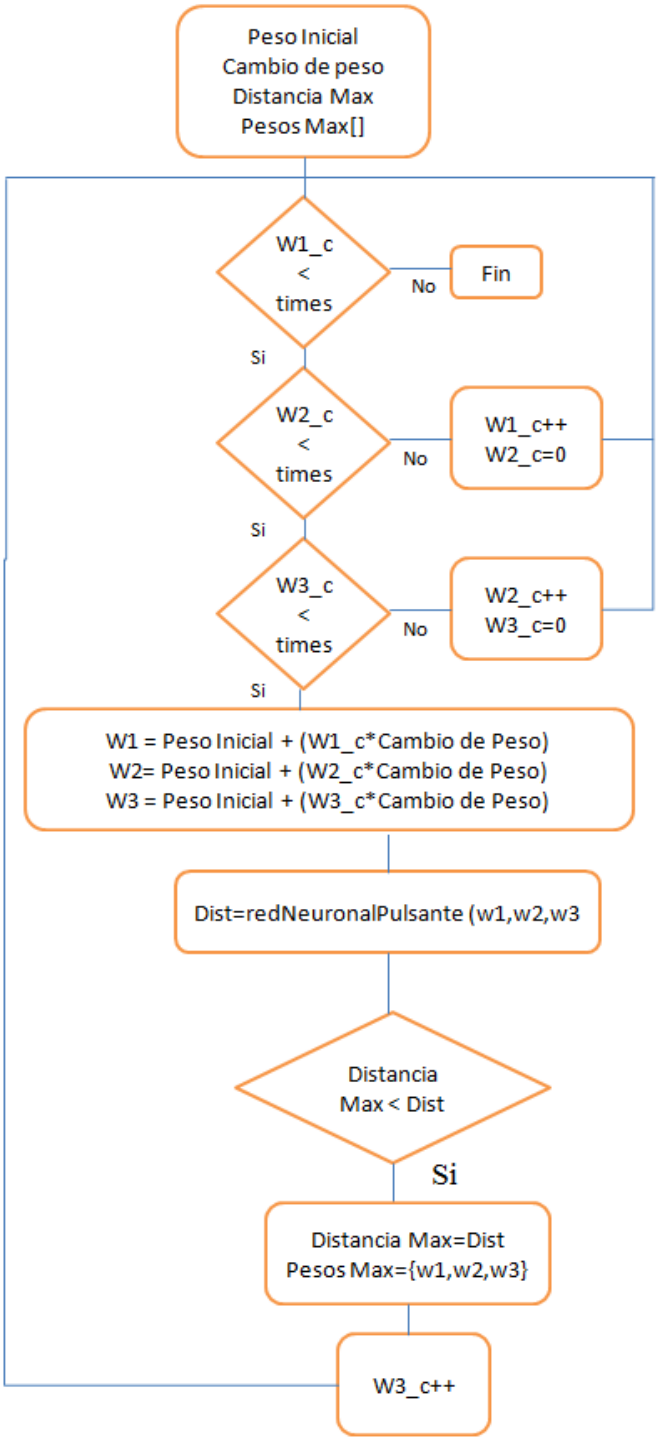


Figura 4.4.1.2.1. Diagrama de flujo del algoritmo de entrenamiento

El algoritmo fue aplicado en el conjunto de entrenamiento descrito en la tabla 4.4.1.1.1. La tabla 4.4.1.2.2 muestra la evolución de los pesos sinápticos durante la ejecución del algoritmo y la distancia mínima existente entre las clases del conjunto de entrenamiento.

Tabla 4.4.1.2.2. Evolución de los pesos sinápticos y la distancia entre clases.

Separación entre clases	W1	W2	W3	W4
0	-295	-495	-495	2905
2	-295	-495	-495	2955
9	-195	-495	-495	2105
34	9	-3	-3	-3
41	5	-3	-3	20
69	**19**	**19**	**-3**	**-3**

Una vez que se alcanzó 69 pulsos de separación entre las clases más cercanas, el algoritmo no encontró una mejor combinación en 20,000 iteraciones y por lo tanto finalizo su ejecución.

Las frecuencias correspondientes a cada vector del conjunto de entrenamiento se guardan en la base de conocimiento y se denominan *frecuencias representantes*. Las frecuencias generadas por cada vector del conjunto de entrenamiento al finalizar la ejecución del algoritmo se muestran en la tabla 4.4.1.2.3.

Tabla 4.4.1.2.3. Frecuencias generadas por el conjunto de entrenamiento después del ajuste pesos sinápticos.

Acción	Frecuencia representante
Avanzar	915
Girar derecha	584
Girar izquierda	842
Lateral derecha	499
Lateral izquierda	999
Tomar objeto	653

La figura 4.4.1.2.3 detalla gráficamente la separación alcanzada entre las clases al finalizar el ajuste de los pesos.

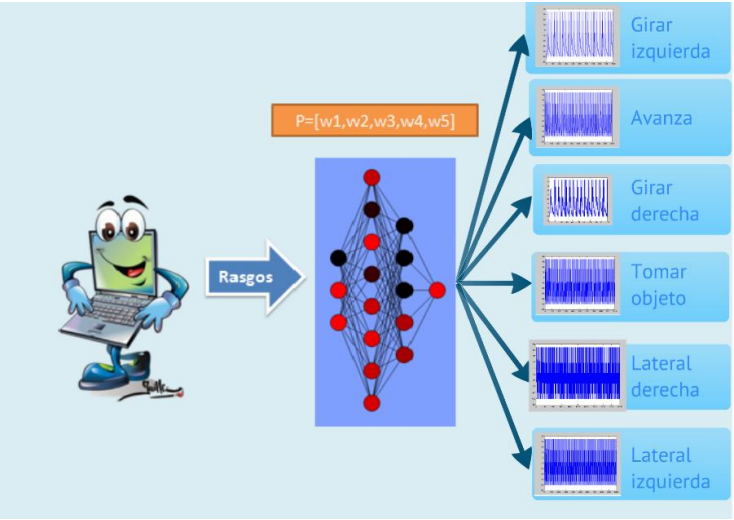


Figura 4.4.1.2.3. Separación gráfica entre las clases.

El apéndice C detalla un análisis de la complejidad del algoritmo.

4.4.1.3 Clasificador euclidiano

Cuando se introduce un nuevo patrón desconocido, se calcula la frecuencia generada por ese patrón y se realiza una clasificación del patrón en base a la menor distancia euclidiana existente entre el la frecuencia del patrón y las frecuencias representantes. La distancia euclidiana se calcula como se muestra a continuación:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

La figura 4.4.1.3.1 detalla la función del clasificador:

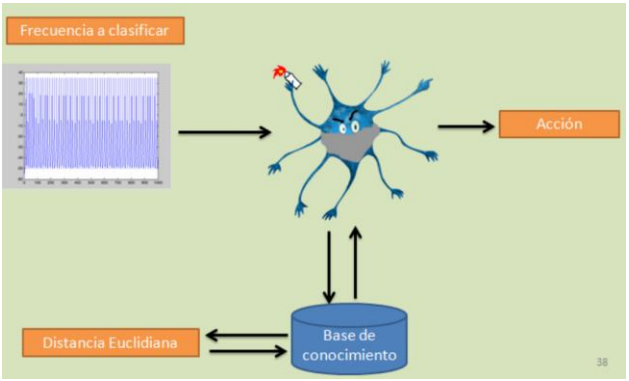


Figura 4.4.1.3.1. Clasificador euclidiano

4.4.2 Sistema difuso.

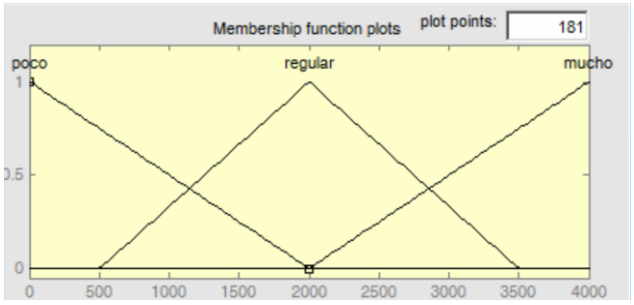
El segundo sub-módulo de la computadora es el sistema difuso, el cual, tiene como objetivo indicar al robot durante cuánto tiempo se debe ejecutar la acción seleccionada por la neurona pulsante.

El sistema difuso, que se encuentra dividido en dos partes, se implementó para las acciones que pueden ser ejecutadas durante un largo período de tiempo tales como *Avanzar, Girar Derecha* y *Girar izquierda.*

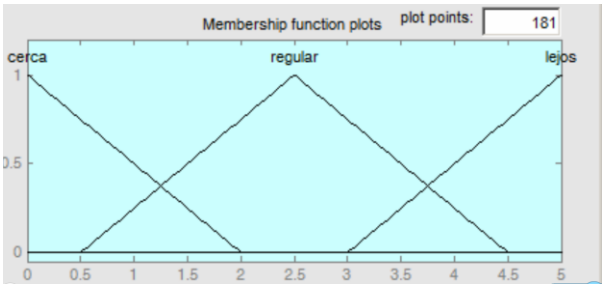
La primer parte del sistema difuso, se enfoca en que el robot humanoide sea capaz de avanzar en dirección al objeto deseado durante la mayor cantidad de tiempo posible sin necesidad de enviar rasgos a la neurona pulsante.

Ya que el sensor de distancia del robot humanoide tan sólo proporciona un cambio significante en los valores si un objeto se encuentra a 30 cm del mismo, la primer parte del sistema difuso, encargada de designar durante cuánto tiempo se ejecuta la acción de *Avanzar,* tiene como entrada el número de píxeles obtenido por el sistema de visión HAVIMO 2.0.

El número de píxeles define los conjuntos antecedentes del sistema, mientras que los segundos que la acción será ejecutada, definen los conjuntos consecuentes tal como se puede apreciar en la figura 4.4.2.1.



(a)



(b)

Figura 4.4.2.1. Conjuntos difusos para la acción de *Avanzar*. a) Conjuntos difusos antecedentes. b) Conjuntos difusos consecuentes.

En la figura 4.4.2.2 podemos apreciar la superficie obtenida y que describe la relación entre píxeles y segundos que se ejecuta la acción.

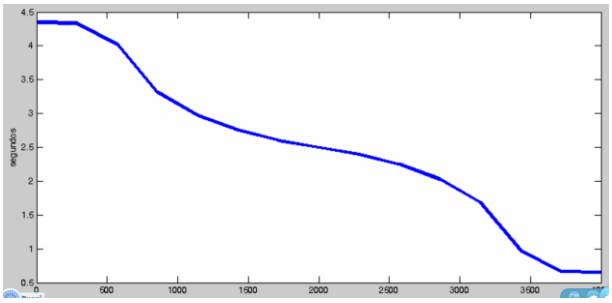


Figura 4.4.2.2. Relación de píxeles con segundos.

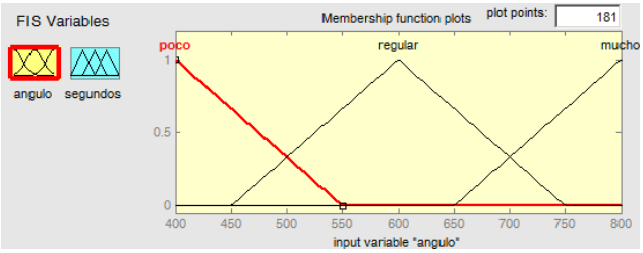
Tal como se puede apreciar en la figura anterior, mientras menor sea el número de píxeles visualizados mayor será el tiempo que el robot avanzará. Esto se debe a que objeto se encuentra a una distancia más lejana. Por otra parte, mientras mayor sea

el número de píxeles, significa que el objeto está más cerca y por lo tanto el tiempo de ejecutar la acción *Avanzar* será menor.

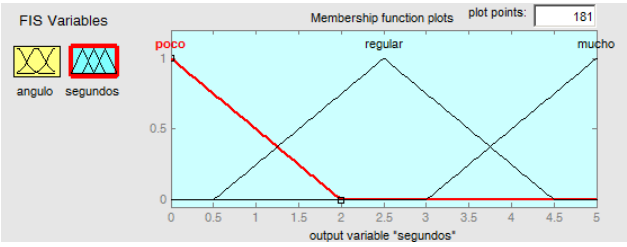
La segunda parte del sistema difuso, se enfoca en que el robot humanoide sea capaz de girar en dirección al objeto deseado tantos segundos como sean necesarios para posicionarse de frente al objeto, sin necesidad de enviar rasgos a la neurona pulsante.

El sistema hace uso de la posición del motor donde está colocada la cámara HAVIMO 2.0, ya que ésta indica en qué medida el objeto se encuentra a la izquierda o a la derecha del robot una vez que se localiza el objeto.

La posición del motor define los conjuntos antecedentes del sistema, mientras que los segundos que la acción será ejecutada definen el conjunto consecuente tal como se puede apreciar en la figura 4.4.2.3.



(a)



(b)

Figura 4.4.2.3. Conjuntos difusos para las acciones “Girar izquierda” y “Girar derecha” a) Conjuntos difusos antecedentes. b) Conjuntos difusos consecuentes.

En la figura 4.4.2.4 podemos apreciar la superficie obtenida, la cual, describe la relación entre la posición del motor y los segundos que se ejecuta la acción.

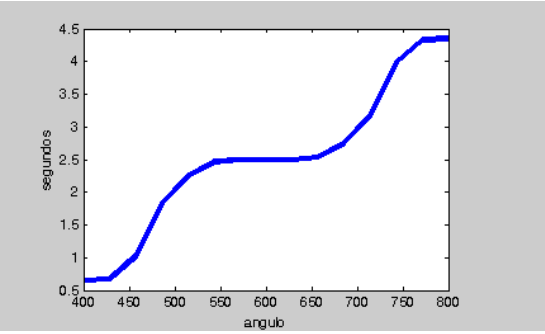


Figura 4.4.2.4. Relación de posición del motor con segundos.

Tal como se puede apreciar en la figura anterior, mientras mayor sea la posición del motor obtenida, mayor será el tiempo que el robot girará ya que el objeto se encuentra más a la izquierda o a la derecha.

Un ejemplo numérico del sistema difuso se muestra en el apéndice B.

4.4.3. Java Speech

El último sub-módulo de la computadora es el reconocedor de comandos verbales. Este sub-módulo tiene como objetivo capturar una instrucción, dada por el usuario mediante un comando de voz, y procesarla para obtener la acción objetivo que el robot realizará.

La gramática utilizada se estableció en base a las diferentes acciones que se pueden realizar tal como se muestra a continuación:

[<dato1><dato2><dato4><dato3>]
[<dato1><dato3><dato4><dato2>]
[<dato5><dato2>]
[<dato5><dato3>]
[<dato6><dato2>]
[<dato6><dato3>]

<dato1>=Llevar Objeto;
<dato2>=Rojo;
<dato3>=Azul;
<dato4>=Hacia Objeto;
<dato5>=Localizar Objeto;
<dato6>=Tomar Objeto;

Con ésta gramática se pueden hacer 6 combinaciones para 2 colores diferentes las cuales son:

1) “Llevar objeto rojo hacia objeto azul”.
2) “Llevar objeto azul hacia objeto rojo”.
3) “Localizar objeto rojo”.
4) “Localizar objeto azul”.
5) “Tomar objeto azul”.
6) “Tomar objeto rojo”.

Sin embargo, la gramática puede ser expandida tanto como sea posible y agregar tantos colores o instrucciones como sea necesario.

Capítulo 5

Resultados

En este capítulo se detallan los resultados obtenidos al aplicar la metodología descrita en el capítulo anterior. Se muestran los experimentos realizados para cada una de las instrucciones permitidas por la gramática de Java Speech y además se muestra la eficiencia por cada una de ellas.

5.1 Experimento: Localizar objeto

Con el objetivo de medir la eficacia del robot para localizar el objeto, mediante el movimiento del motor transversal donde se encuentra colocada la cámara HAVIMO 2.0, el robot y al objeto deseado, fueron colocados en cuatro configuraciones diferentes dentro del espacio de trabajo y se cuantificó el número de ocasiones que la cámara HAVIMO 2.0 identificaba el objeto y enviaba los momentos geométricos calculados a la computadora.

La figura 5.1.1 muestra la configuración 1, en la cual, el objeto se encuentra localizado a 30 cm del robot y está posicionado de frente con el mismo.

Figura 5.1.1. Localización: Configuración 1.

La figura 5.1.2 muestra la configuración 2, en la cual, el objeto se encuentra localizado a 50 cm del robot y está posicionado al costado derecho mismo.

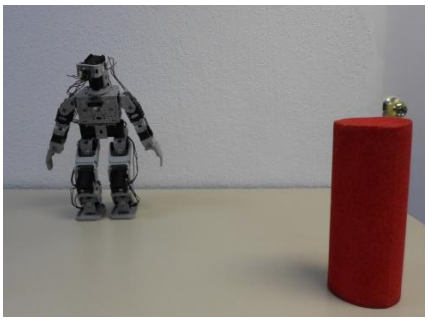


Figura 5.1.2. Localización: Configuración 2.

La figura 5.1.3 muestra la configuración 3, en la cual, el objeto se encuentra localizado a 60 cm del robot y está posicionado al costado izquierdo del mismo.

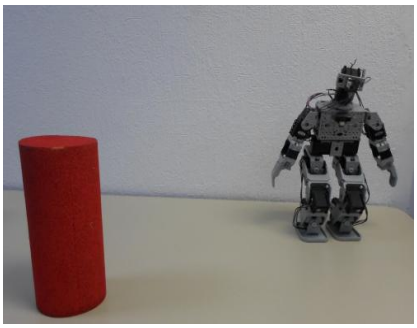


Figura 5.1.3. Localización: Configuración 3.

La figura 5.1.4 muestra la configuración 4, en la cual, el objeto se encuentra localizado a 60 cm del robot y está posicionado detrás del mismo.

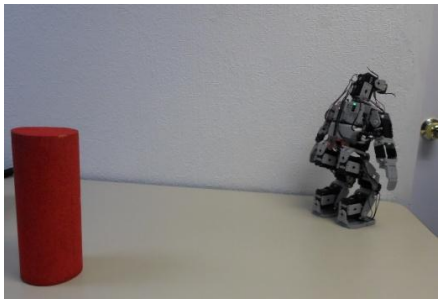


Figura 5.1.4. Localización: Configuración 4.

Es importante mencionar que el movimiento transversal realizado por el motor que tiene montado el sistema HAVIMO 2.0, está restringido a 180° de movilidad, y por lo tanto, en caso de que el robot sea incapaz de encontrar el objeto y el torque

máximo del motor se alcance, entonces es necesario que el robot gire sobre su propio eje tantas veces como sea necesario para poder localizar los objetos que estén fuera del área inicial de búsqueda.

La figura 5.1.5 muestra la gráfica de eficiencia del experimento.

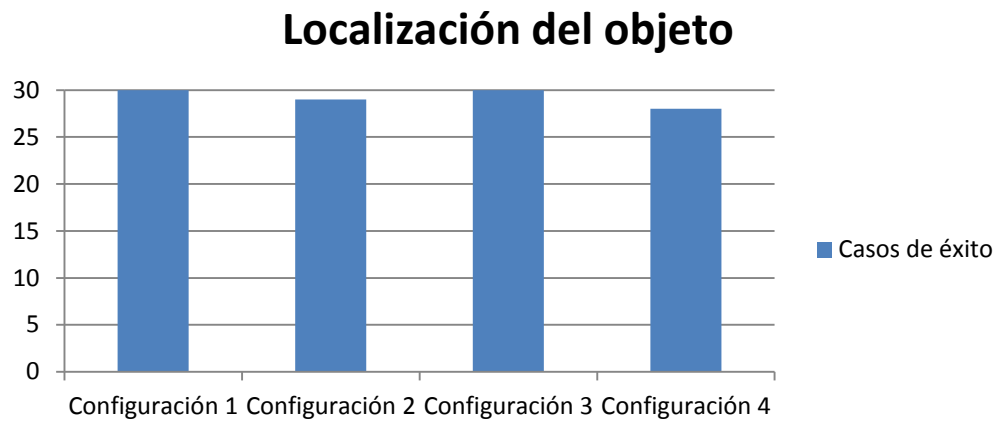


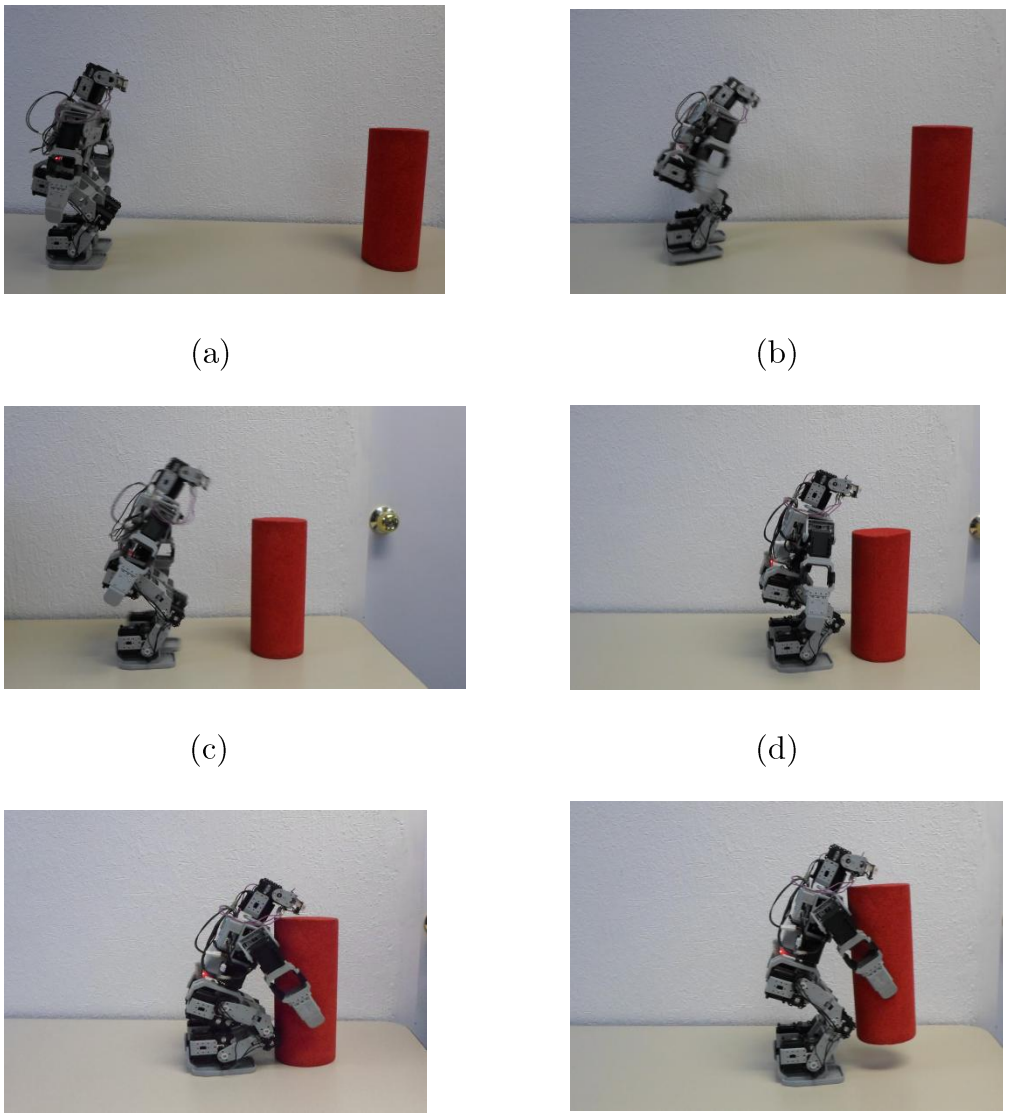
Figura 5.1.5. Casos de éxito donde se localizó el objeto en cada una de las diferentes configuraciones descritas.

Es importante mencionar que, debido al uso de la cámara HAVIMO 2.0, todos los experimentos están relacionados estrechamente con las condiciones de luz. Esta es la razón por la cual existen casos donde no se tuvo éxito. Si las condiciones de luz son idóneas entonces se obtendrá 100% de eficiencia en este experimento.

5.2 Experimento: Tomar objeto

Con el objetivo de medir la eficacia de la neurona pulsante y el sistema difuso, el robot fue puesto a prueba mediante un experimento donde debía ser capaz de desplazarse y tomar el objeto deseado. El robot, fue colocado en diferentes posiciones con respecto al objeto y se cuantificó el número de ocasiones que el robot tomaba el objeto, así como el tiempo en segundos que necesitaba para ejecutar la acción.

La figura 5.2.1 muestra la configuración 1, en la cual, el objeto se encuentra localizado a 30 cm del robot y está posicionado de frente con el mismo. Además se muestra una secuencia de imágenes que muestran la ejecución de la tarea.



(a) (b)

(c) (d)

(e) (f)

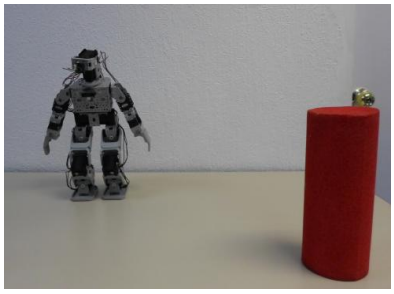
Figura 5.2.1. Tomar objeto: Configuración 1.

La tabla 5.2.1 muestra un ejemplo de las diferentes acciones que el robot realizó y el tiempo que necesitó por cada acción.

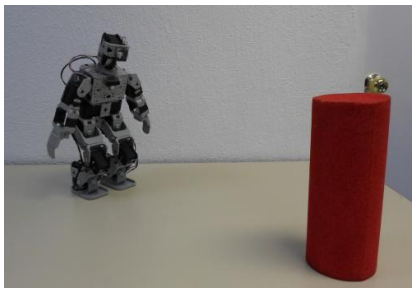
Tabla 5.2.1. Acciones y tiempo requerido para tomar objeto en configuración 1.

Número de acción	**Acción (neurona pulsante)**	**Tiempo en segundos(sistema difuso)**
1	Avanzar	3.0
2	Girar izquierda	2.3
3	Lateral izquierda	-
4	Avanzar	1.6
6	Tomar objeto	-

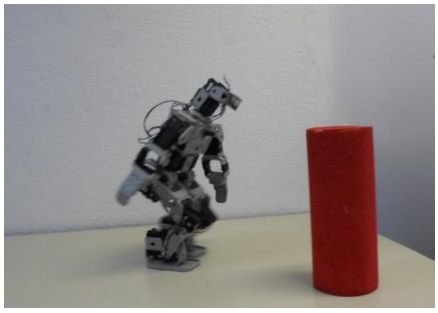
La figura 5.2.2 muestra la configuración 2, en la cual, el objeto se encuentra localizado a 50 cm del robot y está posicionado al costado derecho mismo. Además se muestra una secuencia de imágenes que muestran la ejecución de la tarea.



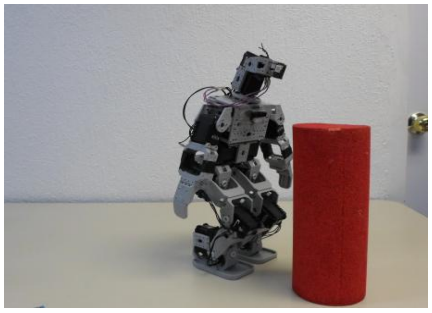
(a)



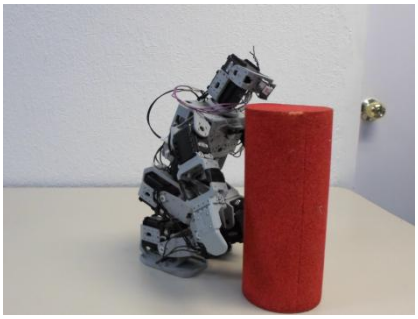
(b)



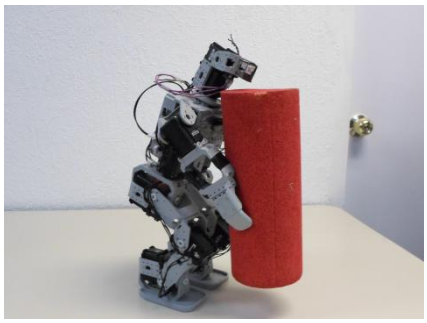
(c)



(d)



(e)



(f)

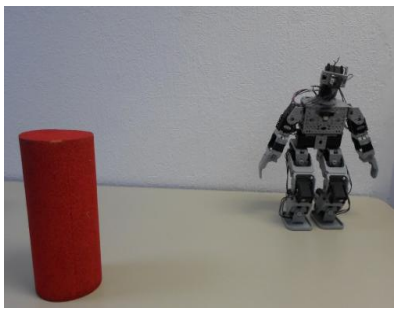
Figura 5.2.2. Tomar objeto: Configuración 2.

La tabla 5.2.2 muestra un ejemplo de las diferentes acciones que el robot realizó y el tiempo que necesitó por cada acción.

Tabla 5.2.2. Acciones y tiempo requerido para tomar objeto en configuración 2.

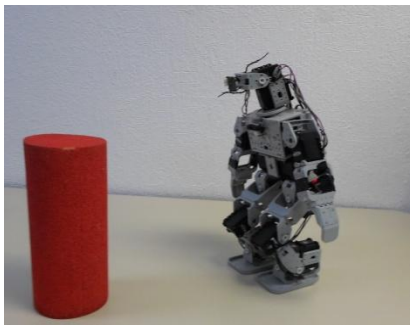
Número de acción	**Acción (neurona pulsante)**	**Tiempo en segundos (sistema difuso)**
1	Girar izquierda	2.6
2	Avanzar	2.2
3	Girar izquierda	2.0
4	Lateral izquierda	-
6	Lateral izquierda	-
7	Tomar objeto	-

La figura 5.2.3 muestra la configuración 3, en la cual, el objeto se encuentra localizado a 60 cm del robot y está posicionado al costado izquierdo del mismo. Además se muestra una secuencia de imágenes que muestran la ejecución de la tarea.

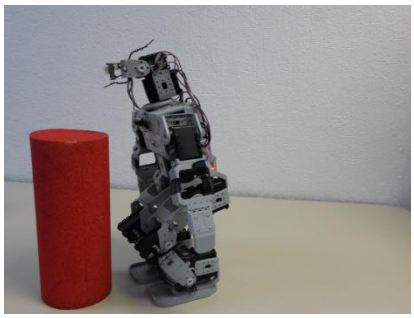


(a)

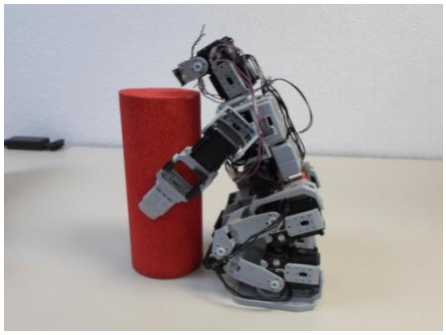
(b)



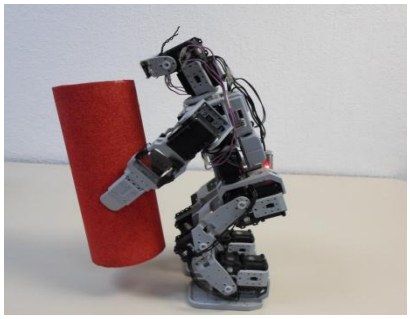
(c)



(d)



(e)



(f)

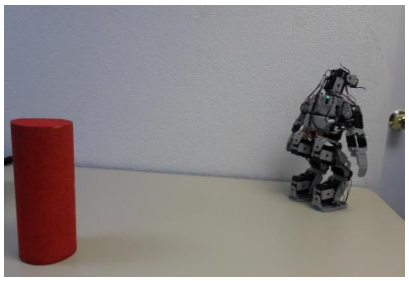
Figura 5.2.3. Tomar objeto: Configuración 3.

La tabla 5.2.3 muestra un ejemplo de las diferentes acciones que el robot realizó y el tiempo que necesitó por cada acción.

Tabla 5.2.3. Acciones y tiempo requerido para tomar objeto en configuración 3.

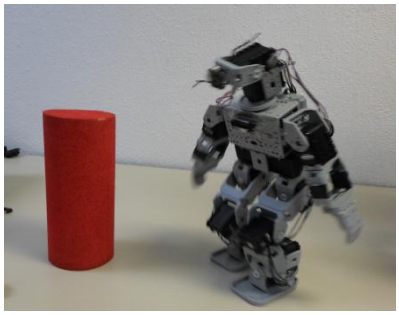
Número de acción	**Acción (neurona pulsante)**	**Tiempo en segundos (sistema difuso)**
1	Girar derecha	2.0
2	Avanzar	2.2
3	Girar izquierda	1.0
4	Girar izquierda	1.0
6	Lateral izquierda	-
7	Lateral derecha	-
8	Tomar objeto	-

La figura 5.2.4 muestra la configuración 4, en la cual, el objeto se encuentra localizado a 60 cm del robot y está posicionado detrás del mismo. Además se muestra una secuencia de imágenes que muestran la ejecución de la tarea.

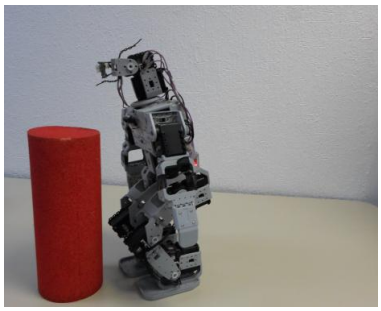


(a)

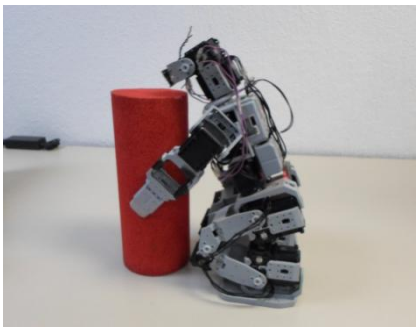
(b)



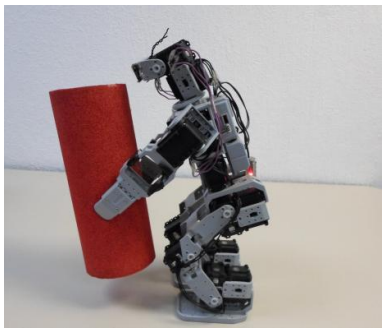
(c)



(d)



(e)



(f)

Figura 5.2.4. Tomar objeto: Configuración 4.

La tabla 5.2.4 muestra un ejemplo de las diferentes acciones que el robot realizó y el tiempo que necesitó por cada acción.

Tabla 5.2.4. Acciones y tiempo requerido para tomar objeto en configuración 4.

Número de acción	**Acción (neurona pulsante)**	**Tiempo en segundos (sistema difuso)**
1	Girar derecha	4.3
2	Girar izquierda	2.0
3	Girar derecha	1.0
4	Avanzar	2.0
6	Girar izquierda	2.0
7	Avanzar	1.2
8	Avanzar	1.1
9	Lateral izquierda	-
10	Tomar objeto	-

La figura 5.2.5 muestra la gráfica de eficiencia del experimento.

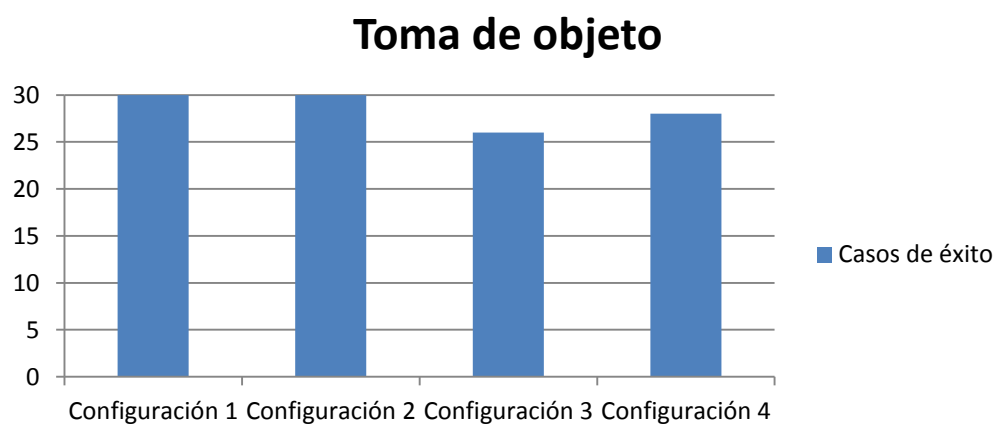


Figura 5.2.5. Casos de éxito donde se tomó el objeto en cada una de las diferentes configuraciones descritas.

Estos experimentos están relacionados estrechamente con las condiciones de luz. Esta es la razón por la cual existen casos donde no se tuvo éxito. En los casos donde no se tuvo éxito, se presentaron perturbaciones del medio ambiente que afectaron el sensor de distancia y algunos valores del sistema de visión HAVIMO 2.0, por lo tanto, el robot tomó la decisión incorrecta y tiró el objeto. A pesar de ello se logró una eficiencia del 90%.

La figura 5.2.6 muestra la gráfica del tiempo necesario para ejecutar la acción.

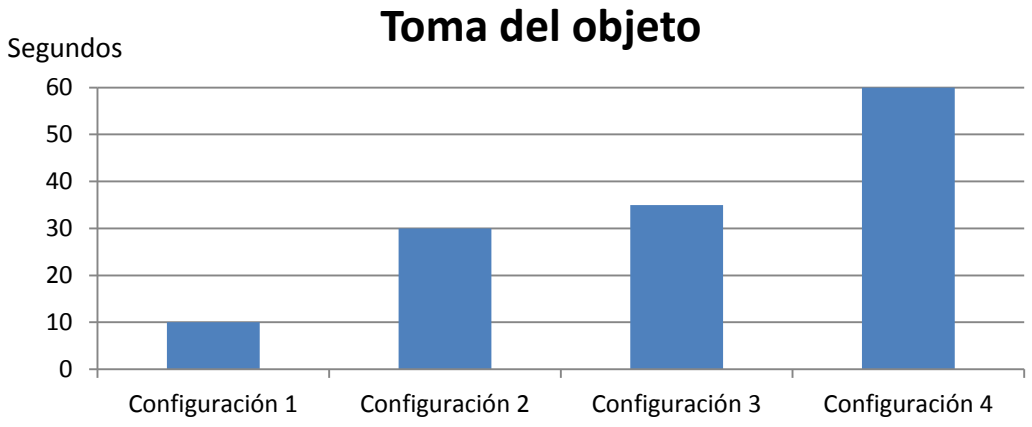


Figura 5.2.6. Tiempo promedio que tardó el robot en tomar el objeto en cada configuración.

.

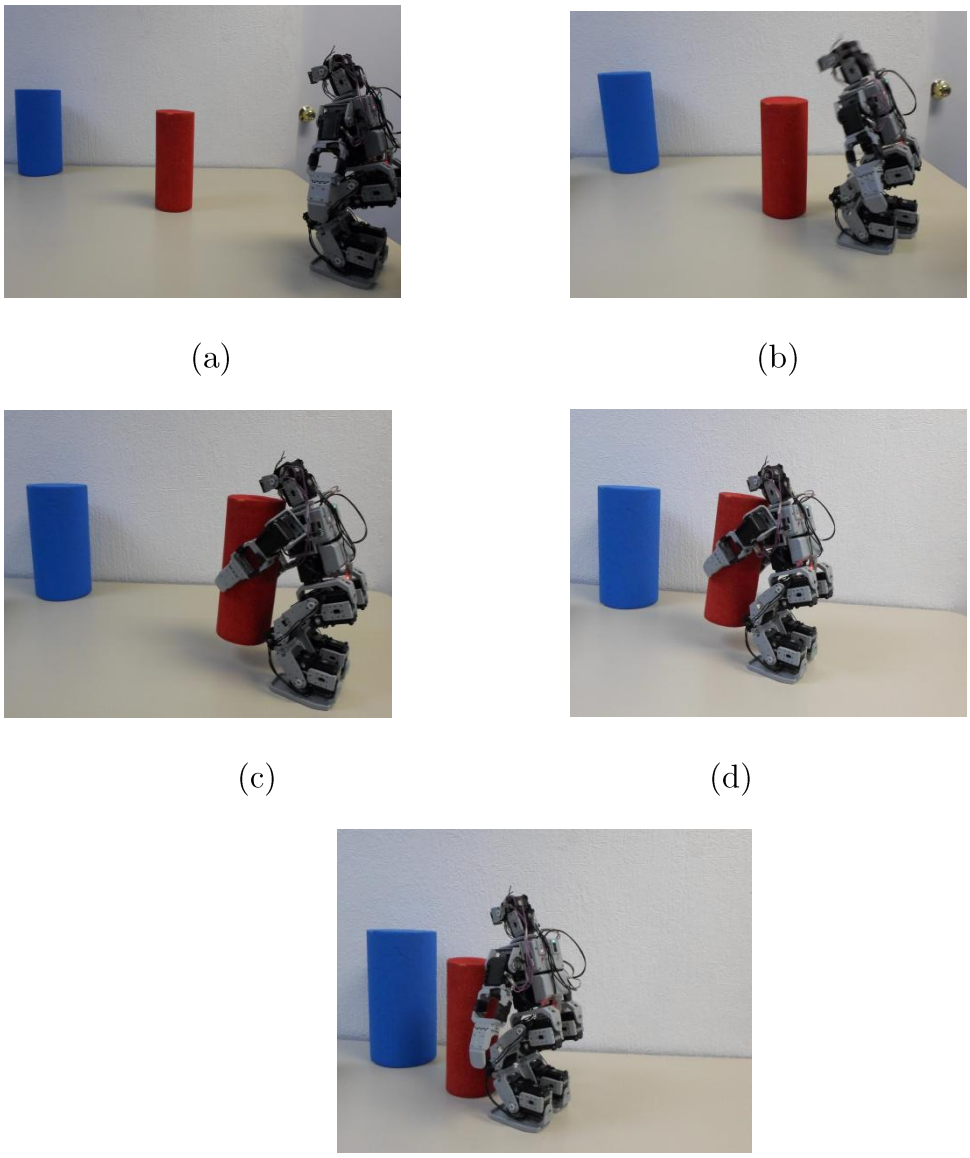
La figura 5.2.6 muestra el tiempo promedio necesario para llevar a cabo la tarea. A partir de éste experimento, se pudo observar que el robot no siempre ejecuta las mismas acciones dada una configuración, esto se debe a que el movimiento mecánico no siempre origina el mismo desplazamiento y además varía dependiendo del material de la superficie en la que se realice el experimiento. Sin embargo, el uso de la neurona pulsante permite que, aunque los movimientos mecánicos no sean exactos, el robot reajuste su posición tomando las decisiones apropiadas para ejecutar el experimento de forma satisfactoria.

5.3 Experimento: Tomar objeto y transportarlo

Una vez completada la tarea de tomar el objeto deseado, el robot debe completar la tarea de transportarlo hasta la posición meta. Para lograr esto, se hizo uso de la misma red neuronal entrenada pero el valor del sensor de distancia se fijó en 50, esto se debe a que el robot humanoide tiene el objeto en sus manos y por lo tanto la lectura del sensor se corrompe. En lugar de eso, se hizo uso del número de píxeles obtenido por el sistema HAVIMO 2.0 para calcular la cercanía con respecto a la posición meta, sin embargo, está medida resulta inexacta.

Con el objetivo de medir la eficacia del sistema, el robot fue puesto a prueba mediante un experimento donde debía ser capaz de desplazarse, tomar el objeto deseado y llevarlo lo más cercano a la posición meta. El robot fue colocado en diferentes posiciones con respecto al objeto deseado y se cuantificó el número de ocasiones que el robot transportó el objeto así como el tiempo requerido en segundos. Este experimento obedeció al comando verbal “Llevar objeto rojo hacia objeto azul”.

La figura 5.3.1 muestra la configuración 1, en la cual, el objeto se encuentra localizado a 30 cm del robot y está posicionado de frente con el mismo, además la posición meta se encuentra a 30 cm del objeto deseado. Además se muestra una secuencia de imágenes que muestran la ejecución de la tarea.



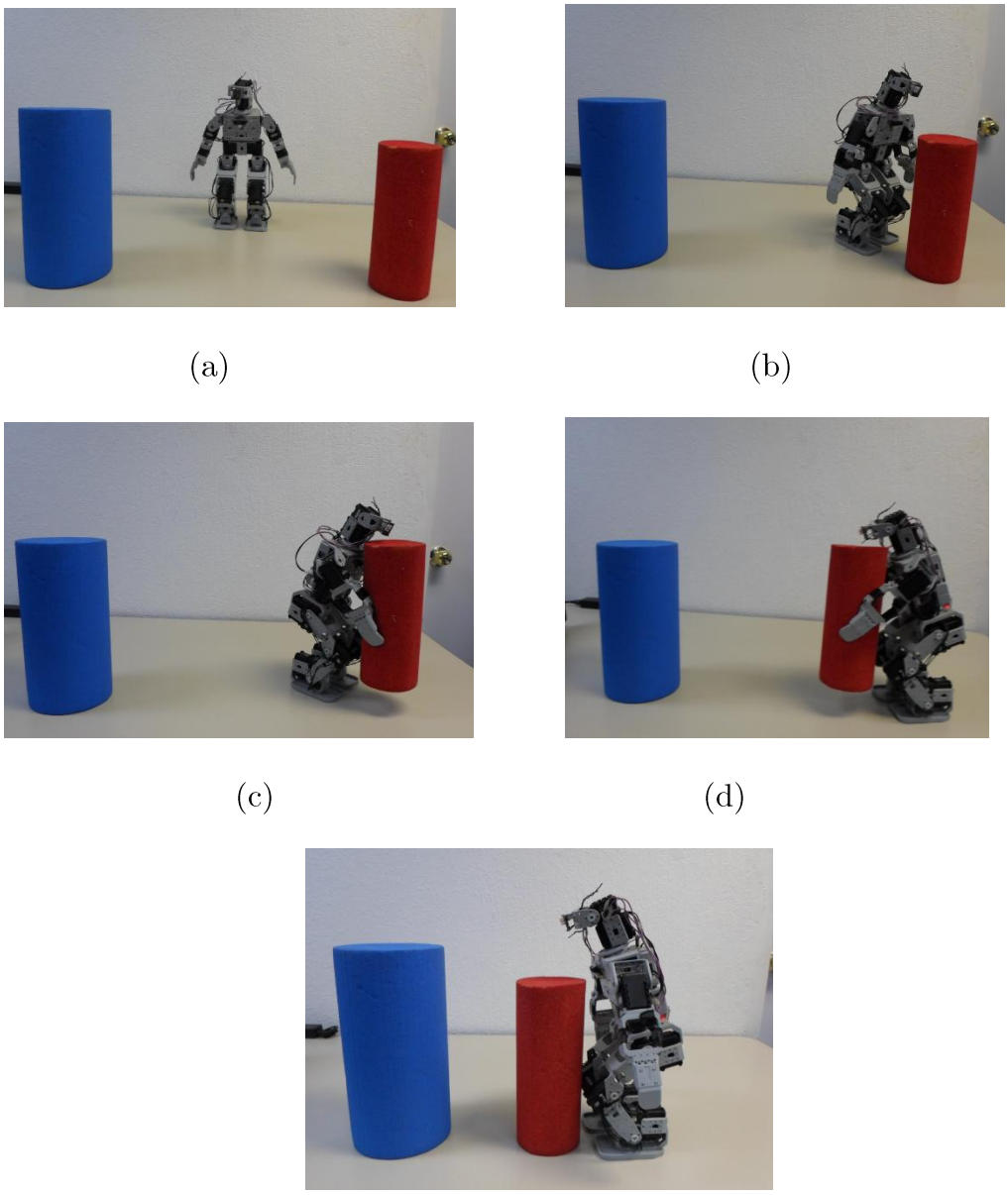
(a) (b)

(c) (d)

(e)

Figura 5.3.1. Transportar objeto: Configuración 1.

La figura 5.3.2 muestra la configuración 2, en la cual, el objeto se encuentra localizado a 30 cm del robot y está posicionado al costado derecho mismo, además la posición meta se encuentra 30 cm del costado izquierdo del robot. Además se muestra una secuencia de imágenes que muestran la ejecución de la tarea.



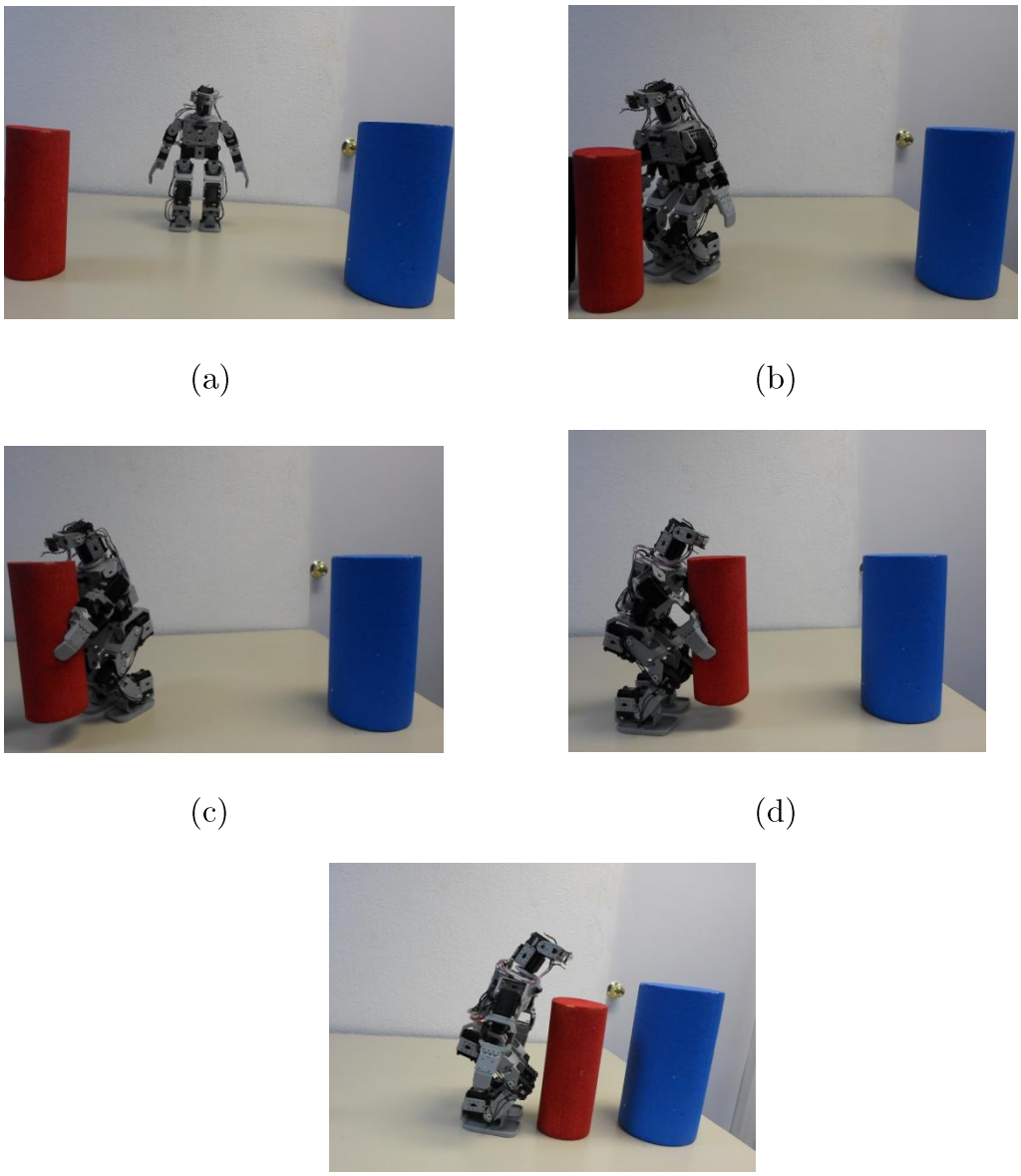
(a) (b)

(c) (d)

(e)

Figura 5.3.2. Transportar objeto: Configuración 2.

La figura 5.3.3 muestra la configuración 3, en la cual, el objeto se encuentra localizado a 30 cm del robot y está posicionado al costado izquierdo del mismo, además la posición meta se localiza 30 cm del costado izquierdo del robot. Además se muestra una secuencia de imágenes que muestran la ejecución de la tarea.



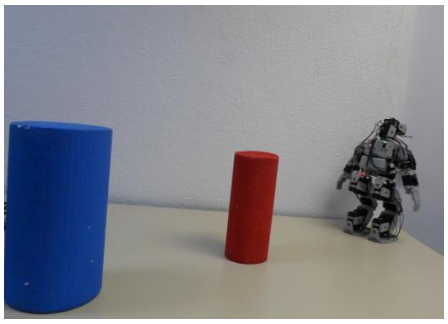
(a) (b)

(c) (d)

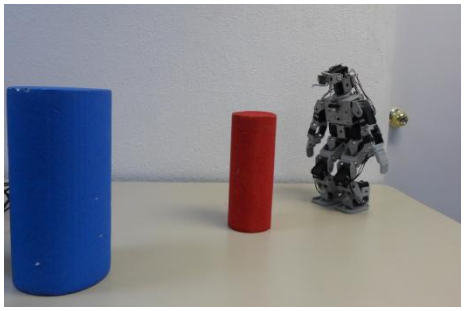
(e)

Figura 5.3.3. Transportar objeto: Configuración 3.

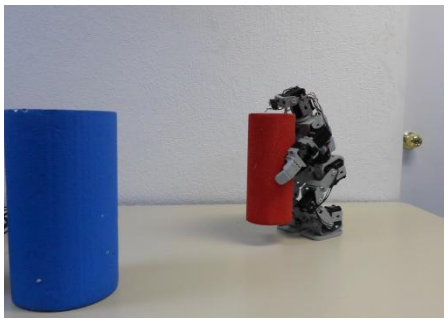
La figura 5.3.4 muestra la configuración 4, en la cual, el objeto se encuentra localizado a 30 cm del robot y está posicionado detrás del mismo, además la posición meta se encuentra a 30 cm detrás del objeto deseado.



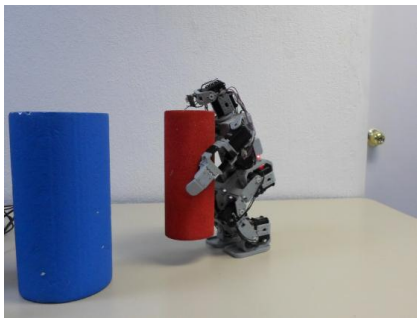
(a)



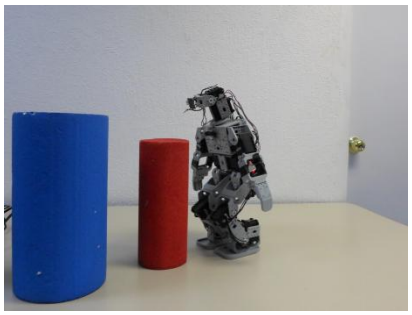
(b)



(c)



(d)



(e)

Figura 5.3.4. Transportar objeto: Configuración 4.

La figura 5.3.5 muestra la gráfica de eficiencia del experimento.

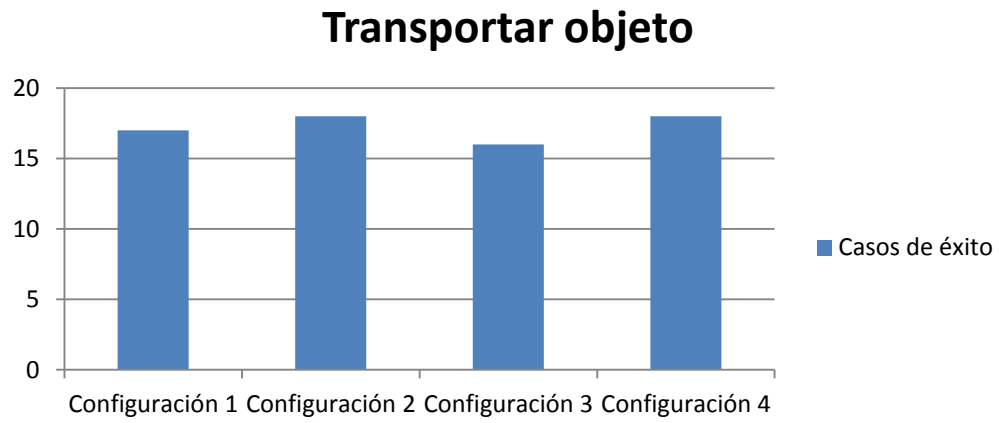


Figura 5.3.5. Casos de éxito donde se transportó el objeto en cada una de las diferentes configuraciones descritas.

Los casos de éxito fueron considerados donde el robot dejaba el objeto en un radio de 30 cm de la posición meta. Dado que se está considerando el número de píxeles para calcular la cercanía de la posición meta, se redujeron los casos de éxito en comparación de la tarea de tomar objeto debido a que el número de píxeles variaba por pequeños cambios de luz. Por otra parte, las perturbaciones mecánicas fueron más comunes durante este experimento debido a que el robot está cargando un objeto y esto afecta la dinámica del humanoide y, por lo tanto, el tiempo se incrementa.

La figura 5.3.6 muestra la gráfica del tiempo necesario para ejecutar la acción.

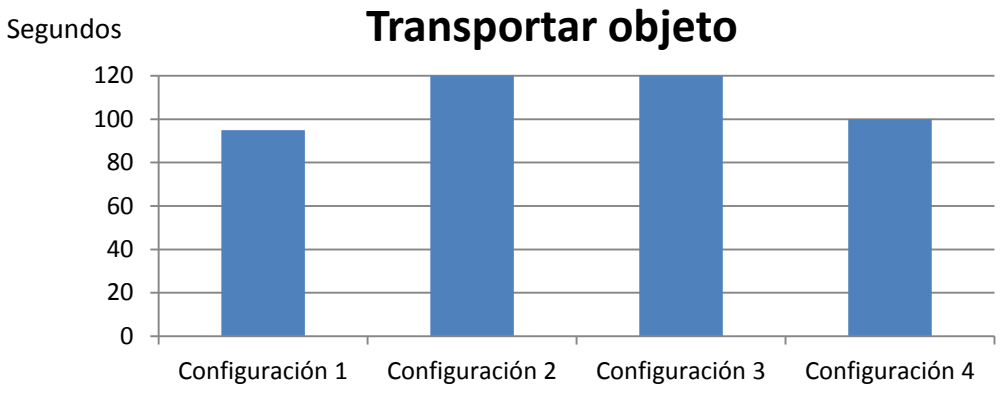


Figura 5.3.6. Tiempo promedio que tardó el robot en transportar el objeto en cada configuración.

Podemos observar que la configuración 2 y 3 requieren mayor tiempo de ejecución, esto debido a que, una vez que se toma el objeto, el robot pierde el ángulo de visión de la posición meta y por lo tanto, es necesario girar cargando el objeto lo que resulta en una mayor cantidad de tiempo. En todas las configuraciones el tiempo se dispara ya que el objeto le hace perder al robot cierta dirección y se desvía constantemente, sin embargo, en un 80% de las ocasiones es capaz de transportarlo hasta la posición meta.

Capítulo 6

Conclusiones y trabajo futuro

Durante este capítulo se detallan las conclusiones que se obtuvieron después de analizar los resultados y además se muestran enfoques que se pueden adoptar para avanzar en este trabajo en el futuro.

6.1 Conclusiones.

A continuación se muestran las conclusiones a las cuales se llegó:

- El uso de neuronas pulsantes en la robótica permiten extender las capacidades del robot para ejecutar tareas de forma autónoma.
- El aprendizaje supervisado permite a un operador transferirle una habilidad a un robot sin necesidad de que el usuario posea conocimientos en programación.
- La combinación de neuronas pulsantes, visión por computadora y sistemas difusos permite tomar a un robot decisiones correctas en circunstancias desconocidas para la tarea de tomar y desplazar un objeto.
- Una tarea ejecutada por un robot que dependa de visión por computadora, depende estrechamente de las condiciones de luz.
- El hacer uso de la cámara HAVIMO 2.0 limita la cantidad de información que podemos obtener del entorno, ya que no podemos acceder a una matriz de píxeles, por el contrario, sólo podemos acceder a los datos procesados por el sistema de visión HAVIMO 2.0.
- Una sola neurona pulsante es capaz de resolver un problema de clasificación para múltiples clases linealmente separables.
- Las neuronas pulsantes permiten transformar problemas lineales de n dimensiones a una sola dimensión, la frecuencia.
- Al realizar el ajuste de los pesos sinápticos, mediante algoritmos exhaustivos, ecuaciones diferenciales o mediante cómputo evolutivo, se separan las clases en el dominio frecuencial lo suficiente para poder realizar una clasificación exitosa para cualquier dato de entrada nuevo.

- El problema de clasificación se simplifica enormemente cuando los valores se encuentran en una sola dimensión.
- La capacidad de solucionar problemas lineales o no lineales mediante una sola neurona pulsante está directamente relacionada con la operación o núcleo utilizado para calcular la corriente de entrada.
- A diferencia de un perceptrón, que por definición utiliza el producto punto, una neurona pulsante puede hacer uso de un núcleo no lineal y resolver problemas no lineales.
- Otra forma de poder resolver problemas no lineales sería incorporar más neuronas y más capas como en el caso de redes neuronales de primera y segunda generación.
- El entrenamiento exhaustivo de neuronas pulsantes tiene una complejidad exponencial y por lo tanto es ineficiente.
- En el caso de una neurona pulsante, rasgos más grandes se necesitan pesos más pequeños para alcanzar más separación entre las clases y viceversa.
- Los sistemas difusos pueden funcionar como un complemento para sistemas de robótica y redes neuronales pulsantes para hacer más eficiente la ejecución de tareas.
- Una neurona pulsante aventaja a un sistema difuso en el aspecto que el usuario debe intervenir más en el sistema difuso para definir los parámetros.
- En lugar del sistema difuso, es posible utilizar otra neurona pulsante para calcular el tiempo de ejecución de las acciones.

6.2 Trabajo futuro.

La presente tesis tiene diferentes enfoques en los cuales puede seguir evolucionando entre los que destacan:

- Implementar el sistema con una neurona pulsante que aprenda en línea, es decir, que haga uso del modelo adaptativo que ofrecen las neuronas pulsantes para agregar nuevos datos y nuevos rasgos al conjunto de entrenamiento y ajustar los pesos mientras aprende a ejecutar la acción de forma satisfactoria.

- Utilizar una neurona pulsante con un núcleo no lineal para calcular el tiempo, en el cual, el robot ejecuta la acción de avanzar en base al número de píxeles en ambientes de iluminación cambiante.
- Utilizar una cámara web que transmita información visual a la computadora e implementar algoritmos de visión, codificados por el usuario, para tener mayor cantidad de información útil.
- Implementar un sistema de pre-procesado de la información que haga uso de algoritmos tales como PCA para poder obtener rasgos relevantes de información obtenida con fuerza bruta. De ésta forma no es necesario que el usuario escoja los rasgos a introducir.
- Desarrollar un algoritmo de entrenamiento de neuronas pulsantes mediante la variación de la operación que calcule la corriente de entrada.
- Desarrollar un algoritmo de entrenamiento para redes neuronales pulsantes multicapa.
- Agregar más sensores que permitan al robot poder hacer una identificación de la posición meta de forma más eficiente y sin depender totalmente de las condiciones de luz.
- Hacer uso de simuladores de redes neuronales pulsantes tales como Ne-Mo para sacar todo el provecho de las redes neuronales y poder controlar sistemas de más grados de libertad y resolver problemas no lineales.

Apéndice A

En éste apéndice se detallan los bloques de código que realizan las principales funciones del sistema.

A.1 Codificación módulo computadora

El siguiente bloque de código detalla la implementación del modelo de Izhikevich en lenguaje C.

```
int Izhikevich(float datos[numRasgos]){

    int i,j=0;
    int C = 100;
    int vr = -60;
    int vt = -40;
    double k = 0.7;
    double a = 0.03;
    int b = -2;
    int c = -50;
    int d = 100;
    int vpeak = 35;
    int T=1000;
    int tau=1;
    const int n=1000;
    double v[n];
    double u[n];
    int numPicos=0;
    int w[numRasgos]={205,-395,5,455};
    int teta =55;
    int r=0;
    double I=0;

    for(i=0;i<n;i++){
        v[i]=vr;
        u[i]=0;
    }
    for(i=0;i<numRasgos;i++)
        I+=datos[i]*w[i];
    I+=teta;
```

```
for(i=0;i<n-1;i++){
    v[i+1]=v[i]+tau*(k*(v[i]-vr)*(v[i]-vt)-u[i]+I)/C;
    u[i+1]=u[i]+tau*a*(b*(v[i]-vr)-u[i]);
    if(v[i+1]>=vpeak){
    numPicos=numPicos+1;
    v[i]=vpeak;
    v[i+1]=c;
    u[i+1]=u[i+1]+d;
}
```

El siguiente bloque de código detalla la implementación del API de Java Speech.

```
import javax.speech.*;
import javax.speech.recognition.*;
import java.io.FileReader;
import java.util.Locale;

public class Escucha extends ResultAdapter {

    static Recognizer recognizer;
    String gst;
    String cadena="";

    @Override
    public void resultAccepted(ResultEvent re) {
        try {
            Result res = (Result) (re.getSource());
            ResultToken tokens[] = res.getBestTokens();
            EscribeFichero ef=new EscribeFichero();

            String args[] = new String[1];
            args[0] = "";
            for (int i = 0; i < tokens.length; i++) {
                gst  = tokens[i].getSpokenText();
                args[0] += gst + " ";
                System.out.print(gst + " ");
                cadena+=gst + " ";
            }
            ef.escribir(cadena);
            recognizer.deallocate();
            args[0] = "Hasta la proxima Cmop!";
            System.out.println(args[0]);
            Lee.main(args);
        } catch (Exception ex) {
            System.out.println("Ha ocurrido algo inesperado " + ex);
        }
    }
```

```
    public static void main(String args[]) {
        try {
            recognizer = Central.createRecognizer(new EngineModeDesc(Locale.ROOT));
            recognizer.allocate();
            FileReader grammar1 = new FileReader("C:/Users/Lulu/Documents/Gramatica.txt");
            RuleGrammar rg = recognizer.loadJSGF(grammar1);
            rg.setEnabled(true);
            recognizer.addResultListener(new Escucha());
            System.out.println("Empieze Dictado");
            recognizer.commitChanges();
            recognizer.requestFocus();
            recognizer.resume();
        } catch (Exception e) {
            System.out.println("Exception en " + e.toString());
            e.printStackTrace();
            System.exit(0);
        }
    }
}
```

A.2 Codificación del módulo robot

La figura A.1 muestra el bloque de código para accesar a la cámara HAVIMO 2.0.

```
START PROGRAM
{
    CamID  = 100
    Color  = 1
    ID[CamID]: ADDR[0(b)]  = 0
    Timer  = 0.256sec
    LOOP WHILE  ( Timer  >  0.000sec  )
    {
    }
    CALL Get_Bounding_Box
    Cx  = Minx  +  Maxx
    Cy  = Miny  +  Maxy
    Cx  = Cx  /  2
    Cy  = Cy  /  2
}
```

(a)

```
FUNCTION Get_Bounding_Box
{
    Max = 0
    LOOP FOR ( Index = 1 ~ 15 )
    {
        Addr = Index * 16
        IF ( ID[CamID]: ADDR[Addr(b)] != 0 )
        {
            Addr = Addr + 1
            IF ( ID[CamID]: ADDR[Addr(b)] == Color )
            {
                Addr = Addr + 1
                Size = ID[CamID]: ADDR[Addr(w)]
                IF ( Size > Max )
                {
                    Max = Size
                    MaxAddr = Addr
                }
            }
```

(b)

```
            }
        }
    }
    IF ( Max == 0 )
    {
        RETURN
    }
    Addr = MaxAddr + 10
    Maxx = ID[CamID]: ADDR[Addr(b)]
    Addr = Addr + 1
    Minx = ID[CamID]: ADDR[Addr(b)]
    Addr = Addr + 1
    Maxy = ID[CamID]: ADDR[Addr(b)]
    Addr = Addr + 1
    Miny = ID[CamID]: ADDR[Addr(b)]
}
```

(c)

Figura A.2.1. Acceso a la cámara HAVIMO 2.0. (a) Configuración de la cámara. (b) Función Get Bounding Box. (c) Acceso a los datos de la cámara.

Una vez que se han accedido a los datos de la cámara HAVIMO 2.0, el módulo robot debe ser capaz de enviar datos de los datos leídos y recibir las acciones a realizar. La figura A.2.2. muestra el envío y la recepción de datos mediante el módulo Zigbee.

```
    sensorDistancia = PORT[6]
}
Remocon TXD = Cx
Remocon TXD = Cy
Remocon TXD = ID[5]: Present position
Remocon TXD = sensorDistancia
Remocon TXD = Max
LOOP WHILE ( respuesta == FALSE )
{
    IF ( Remocon Arrived == TRUE )
    {
        respuesta = TRUE
        recibido = Remocon RXD
```

Figura A.2.2. Envío y recepción de datos.

Es importante mencionar que mientras que se envían varios rasgos solo se recibe una acción a realizar y el tiempo que se realizará. La figura A.2.3 muestra un ejemplo de cómo el código discrimina entre la acción recibida, su ejecución y el tiempo en el cual se realiza.

```
IF ( recibido == R )
{
    WalkCommand = 1
    CALL WalkExecute
    Motion Page = 32
    Timer = 1.280sec * segs
    LOOP WHILE ( Timer > 0.000sec )
    {
        IF ( PORT[6] >= 400 )
        {
            WalkCommand = 0
            CALL WalkExecute
        }
    }
```

Figura A.2.3. Ejecución de la acción.

Apéndice B

En este apéndice se detalla un ejemplo numérico acerca del entrenamiento de la neurona pulsante mediante el algoritmo propuesto y uno del sistema difuso utilizado.

B.1 Ejemplo red neuronal

A continuación se muestra un ejemplo numérico para mostrar el funcionamiento del algoritmo de entrenamiento exhaustivo. Para el ejemplo, se usaron 2 clases linealmente separables con 2 rasgos cada una.

Clase 1: {100,100}

Clase 2: {1,1}

Peso Inicial: -3.95

Cambio: .3

Iteración	**Separación entre clases**	**W1**	**W2**
16	2	0.850000	-3.950000
20	3	2.050000	-3.950000
27	6	4.150000	-3.950000
28	12	4.450000	-3.950000
29	19	4.750000	-3.950000
30	25	5.050000	-3.950000
31	31	5.350000	-3.950000

32	37	5.650001	-3.950000
33	42	5.950000	-3.950000
34	46	6.250000	-3.950000
35	51	6.550000	-3.950000
36	55	6.850000	-3.950000
37	59	7.150001	-3.950000
38	63	7.450000	-3.950000
39	66	7.750000	-3.950000
40	70	8.050000	-3.950000
41	74	8.350000	-3.950000
42	77	8.650000	-3.950000
43	81	8.950000	-3.950000
44	84	9.250000	-3.950000
45	88	9.550000	-3.950000
46	89	9.850000	-3.950000
47	95	10.15000	-3.950000
48	96	10.45000	-3.950000
49	97	10.75000	-3.950000

Una vez que se alcanza la separación de 97 pulsos entre las clases, el algoritmo exploró 20 000 combinaciones más y, dado que no se mejora el resultado obtenido, entonces el algoritmo se detuvo y arrojó los pesos finales que ofrecen una separación de 97 pulsos entre clases, los cuales, son: w1=10.75 y w2=-3.95.

B.2 Ejemplo numérico módulo difuso

Con el objetivo de mostrar el funcionamiento del sistema difuso mediante un ejemplo numérico este se implementó en el toolbox fuzzy de MATLAB.

El sistema difuso consta de 2 partes, la primera consiste en una relación entre los píxeles vistos por la cámara HAVIMO 2.0 y los segundos que el robot avanzará.

La figura B.2.1 muestra cómo se comporta el sistema cuando se introducen 817 píxeles en el sistema.

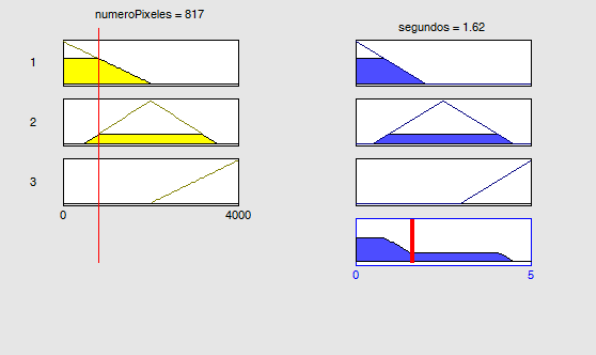


Figura B.2.1. Salida del sistema difuso con 817 píxeles de entrada.

Tal como se puede apreciar en la figura anterior, con el número de píxeles de entrada se obtienen 1.62 segundos, tiempo durante el cual, el robot avanzará hacia el objeto. La figura B.2.2 cómo al ingresar 2427 píxeles al sistema se obtienen 2.67 segundos.

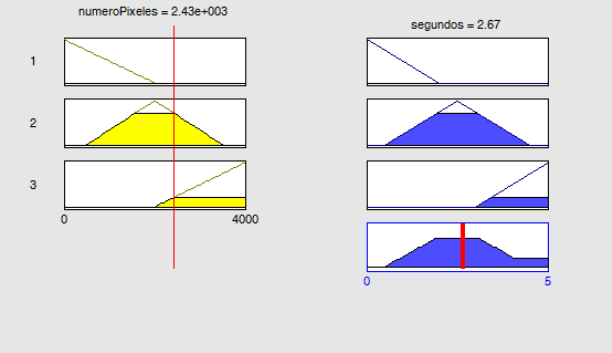


Figura B.2.2. Salida del sistema difuso con 2427 píxeles de entrada.

La segunda parte del sistema difuso consiste en la relación entre la posición del motor y el tiempo, en el cual, el robot realizará el giro a la izquierda o a la derecha. La figura B.2.3 detalla un ejemplo donde la posición del motor tiene un valor de 684.

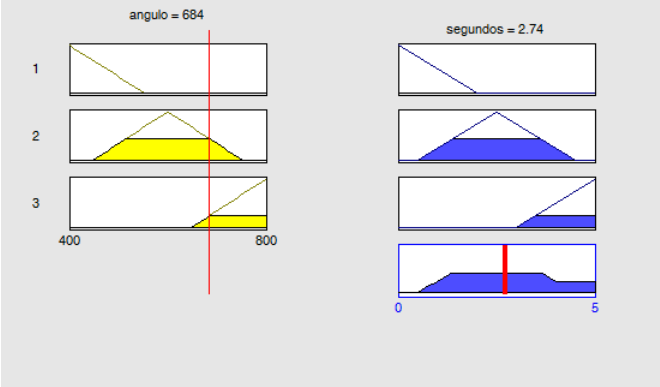


Figura B.2.3. Salida del sistema difuso con el motor en 684 grados.

Apéndice C

C.1 Desempeño del algoritmo de entrenamiento

El desempeño del algoritmo exhaustivo de entrenamiento para neuronas pulsantes, fue evaluado mediante la variación de los parámetros de entrada. Primeramente se varió el número de rasgos por patrón de de entrada, después el número de clases y finalmente ambos.

La figura C.1.1 muestra una gráfica que midió la eficiencia del algoritmo al variar el número de rasgos entre 2 clases.

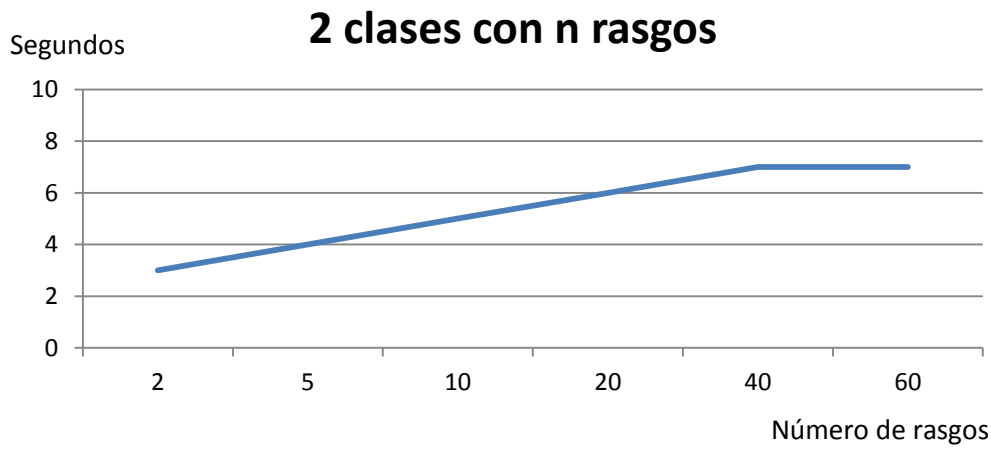


Figura C.1.1. Medición de desempeño: variación del número de rasgos en 2 clases.

Tal cómo se puede apreciar en la figura C.1 a medida que el número de rasgos crece, en el caso donde se tienen 2 clases, el tiempo requerido encuentra su cota asintótica en 7 segundos aproximadamente. Esto se debe a que se estableció como separación máxima 999 y, por lo tanto, en el momento en el que se alcanza dicha separación, el algoritmo se detiene. Sin embargo, si se deseará alcanzar una separación mayor entre las clases entonces el tiempo de entrenamiento crecería también.

La figura C.1.2 muestra una gráfica que midió la eficiencia del algoritmo al variar el número de clases con 2 rasgos.

Figura C.1.2. Medición de desempeño: variación del número de clases con 2 rasgos.

Como podemos apreciar en la figura C.2, a diferencia del número de rasgos, el número de clases aumenta el tiempo de ejecución del algoritmo. Es importante mencionar que, cuando se probó el algoritmo con solo 2 rasgos y n clases, la separación máxima alcanzada por el algoritmo fue de 2 o 3 pulsos ya que hacían falta más rasgos para poder discriminar entre las clases, es decir, las clases no eran linealmente separables.

La figura C.1.3 muestra una gráfica que midió la eficiencia del algoritmo al variar el número de rasgos y el número de clases.

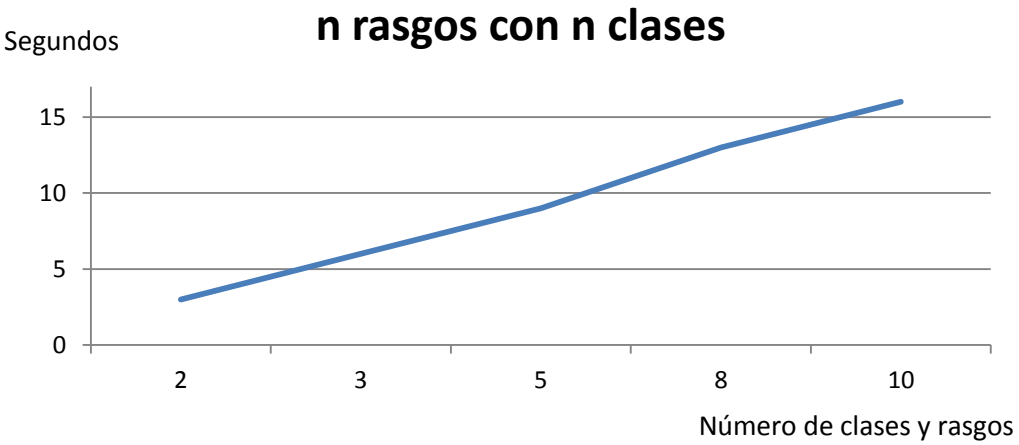


Figura C.1.3. Medición de desempeño: variación del número de rasgos y clases.

En la figura C.1.3 se pudo observar que, al realizar una variación del número de rasgos y el número de clases, el sistema es capaz de discriminar entre las clases ya es posible separar las clases linealmente. Por otro lado, el tiempo en el cual se encuentra la separación establecida por el usuario es similar al primer caso de 2 rasgos y 2 clases.

Apéndice D

En este apéndice se muestra un estudio de las capacidades de un perceptrón de segunda generación para resolver problemas de clasificación para el caso de 3 o mas clases para 2 rasgos.

Dado que las neuronas de primera generación (TLU) utilizan la función escalón para determinar su salida, tienen una salida binaria y, por lo tanto, sin importar el número de rasgos, una sola neurona TLU no tiene la capacidad de poder resolver problemas de clasificación multiclase.

Por otra parte, por definición, una neurona de segunda generación, también llamada perceptrón, tiene como salida la función sigmoide o la función rampa como función de activación y se establece un umbral, por lo general de 0.5, para poder clasificar entre 2 clases. Geométricamente un solo perceptrón traza un hiperplano y es por eso que se considera que tan sólo puede discriminar entre 2 clases diferentes. Sin embargo, al realizar el producto punto entre los rasgos de entrada y los pesos sinápticos y al sumarle un *bias*, obtenemos un valor escalar, el cual, puede ser utilizado, desde un punto de vista algebraico, para realizar una clasificación para problemas multiclase. Dentro de este apartado mostramos una serie de experimentos para resolver este tipo de problemas con un perceptrón.

D. 1 Problemas linealmente separable

La figura D.1.1 muestra la distribución de 3 clases, linealmente separables con 1000 datos por cada clase y con una desviación estándar de 6 unidades entre datos de la clase.

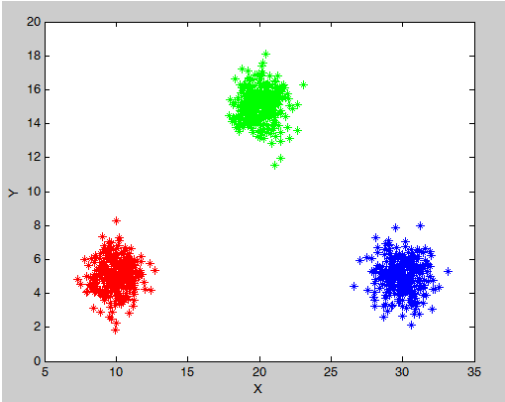


Figura D.1.1. Problema de clasificación de 3 clases con 2 rasgos.

La figura D.1.1 muestra tres clases que no pueden ser separadas geométricamente mediante un hiperplano, sin embargo, desde el punto de vista algebraico esto sí es posible. La topología del perceptrón utilizada se muestra en la figura D.1.2.

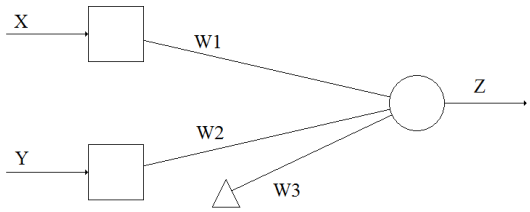


Figura D.1.2. Topología utilizada para probar la capacidad del perceptrón.

Donde:

X y Y definen los valores del patrón de entrada.
W_1 y W_2 definen los pesos sinápticos.
W_3 define el *bias*.
Z define la salida.

Mediante una selección aleatoria de los pesos iniciales, el perceptrón fue entrenado con el algoritmo de retro propagación y se graficó el valor escalar generado por cada entrada del conjunto de entrenamiento y los pesos sintonizados.

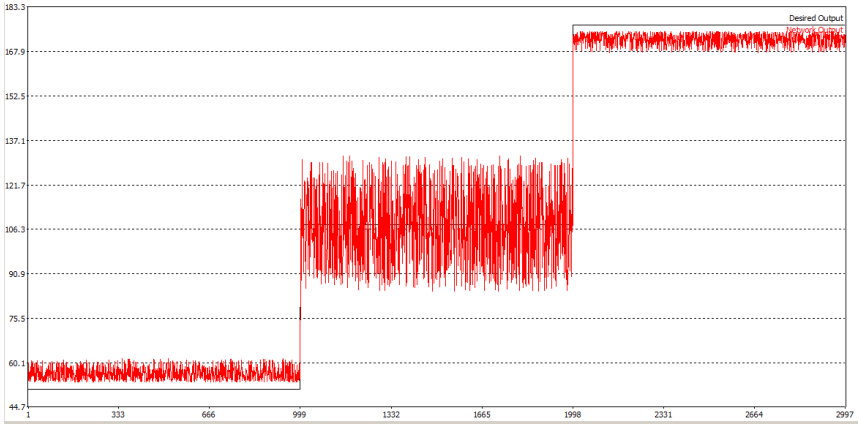
La figura D.1.3 muestra la gráfica generada por el valor escalar correspondiente a cada entrada del conjunto de entrenamiento, al utilizar una función SIGMOIDE de activación.

Tal cómo podemos apreciar en la figura D.1.3, si tenemos p clases linealmente separables en el conjunto de entrenamiento, un solo perceptrón genera p rangos que permiten discriminar entre las p clases.

Por otra parte, la figura D.1.4 muestra la gráfica generada por el valor escalar correspondiente a cada entrada del conjunto de entrenamiento, al utilizar una función RAMPA de activación.

Tal cómo se puede apreciar en la figura D.1.4, al igual que en el caso de la sigmoide, si se utiliza una función rampa, el perceptrón genera p rangos que permiten discriminar entre las p clases, sin embargo, con la función rampa se generan p rangos donde la varianza entre los datos generada por cada clase es la misma, es decir, al observar el gráfico D.1.3 se puede notar que la clase 1 y la clase 3, que se encuentran en los extremos, tienen un rango de números más pequeño debido a la naturaleza geométrica de la función sigmoide, sin embargo, en el caso del gráfico D.1.4, la clase 1 y la 3, que se encuentran en los extremos, los rangos tienen el mismo tamaño. Esto se debe a la forma geométrica de la función rampa.

Valor escalar al utilizar la función sigmoide



Índice del conjunto de entrenamiento

Figura D.1.3. Salida escalar correspondiente a cada entrada del conjunto de entrenamiento para 3 clases y 2 rasgos.

Valor escalar al utilizar la función rampa

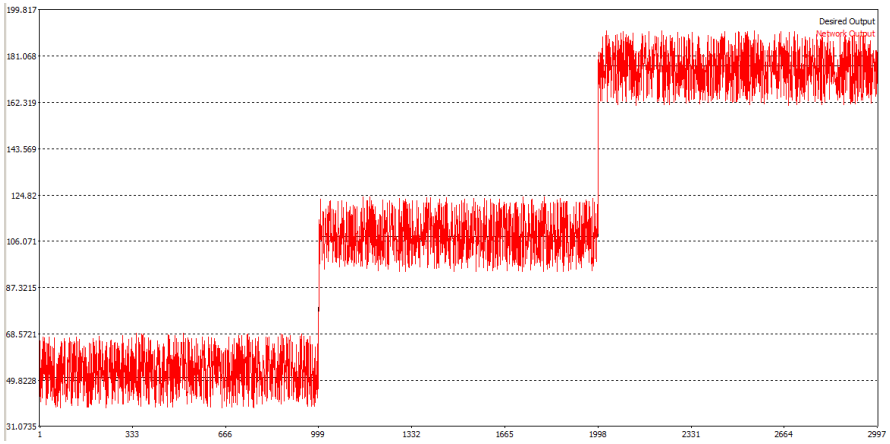


Figura D.1.4. Salida escalar correspondiente a cada entrada del conjunto de entrenamiento para 3 clases y 2 rasgos.

Con el objetivo de mostrar un panorama más amplio del estudio anterior, la topología mostrada en la figura D.1.2 fue utilizada nuevamente pero, ahora, para resolver un problema de 5 clases, linealmente separables con 1000 datos cada una y con una desviación estándar de 6 unidades entre datos de la misma clase tal como se aprecia en la figura D.1.5.

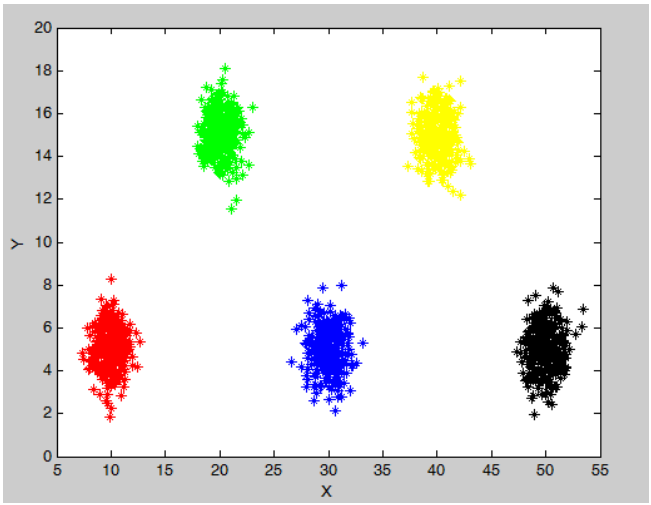


Figura D.1.5. Problema de clasificación de 5 clases linealmente separables con 2 rasgos.

Mediante una selección aleatoria de los pesos iniciales, el perceptrón fue entrenado mediante el algoritmo de retro propagación y se graficó el valor escalar generado por cada entrada del conjunto de entrenamiento y los pesos sintonizados.

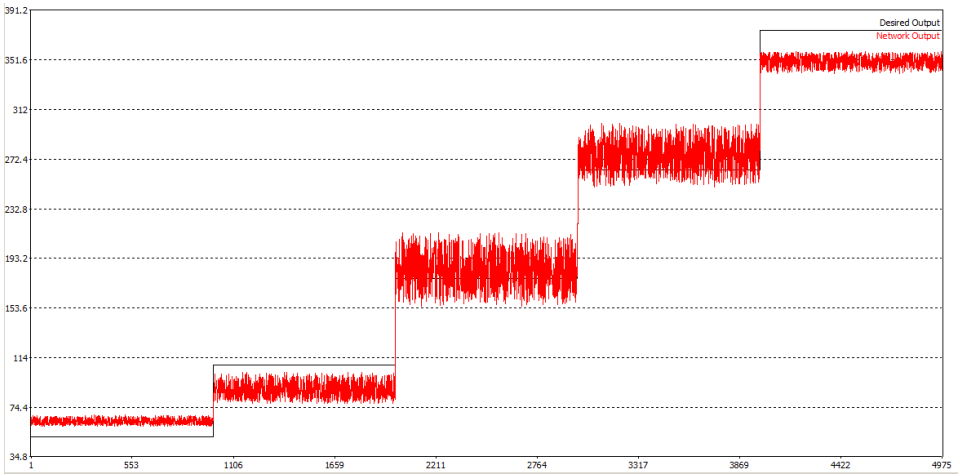
La figura D.1.6 muestra la gráfica generada por el valor escalar correspondiente a cada entrada del conjunto de entrenamiento, al utilizar una función SIGMOIDE de activación.

Dado que tenemos 5 clases, en este caso, se generan 5 rangos que permiten discriminar entre las 5 clases.

Por otra parte, la figura D.1.7 muestra la gráfica generada por el valor escalar correspondiente a cada entrada del conjunto de entrenamiento, al utilizar una función RAMPA de activación.

De igual forma, el perceptrón genera 5 rangos que permiten discriminar entre las 5 clases, sin embargo, podemos notar, que en el gráfico D.1.6, la clase 1 y la clase 5, que se encuentran en los extremos, tienen un rango de números más pequeño debido a la naturaleza geométrica de la función sigmoide, sin embargo, en el caso de la función rampa, la clase 1 y la 5, que se encuentran en los extremos, los rangos tienen el mismo tamaño.

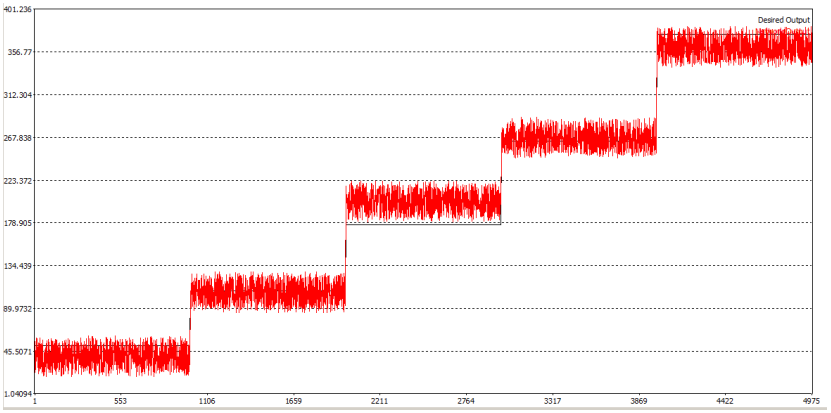
Valor escalar al utilizar la función sigmoide



Índice del conjunto de entrenamiento

Figura D.1.6. Salida escalar de cada entrada del conjunto de entrenamiento para 5 clases y 2 rasgos.

Valor escalar al utilizar la función rampa



Índice del conjunto de entrenamiento

Figura D.1.7. Salida escalar de cada entrada del conjunto de entrenamiento para 5 clases y 2 rasgos.

A través de ambos ejemplos, se ha mostrado, como un solo perceptrón puede resolver un problema de clasificación multiclase si las clases son linealmente separables. A continuación se presenta un ejemplo donde las clases no son linealmente separables y el perceptron no puede resolver el problema.

D. 2 Problema linealmente no separable

La figura D.2.1 muestra la distribución de 2 clases, linealmente no separables con 1000 datos por cada clase y con una desviación estándar de 6 unidades entre datos de la misma clase.

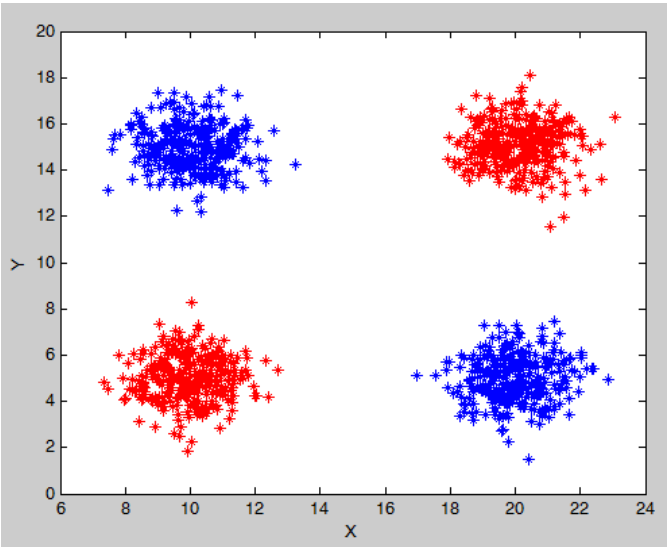


Figura D.2.1. Problema de clasificación de 2 clases linealmente no separables con 2 rasgos.

La topología del perceptrón utilizada se muestra en la figura D.2.1.

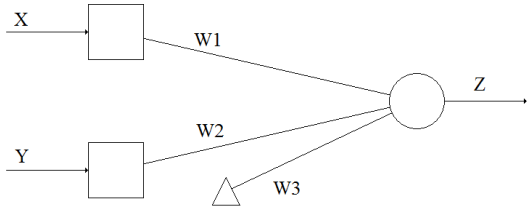


Figura D.2.2. Topología utilizada para probar la capacidad del perceptrón.

Mediante una selección aleatoria de los pesos iniciales, el perceptrón fue entrenado con el algoritmo de retro propagación y se graficó el valor escalar generado por cada entrada del conjunto de entrenamiento y los pesos sintonizados.

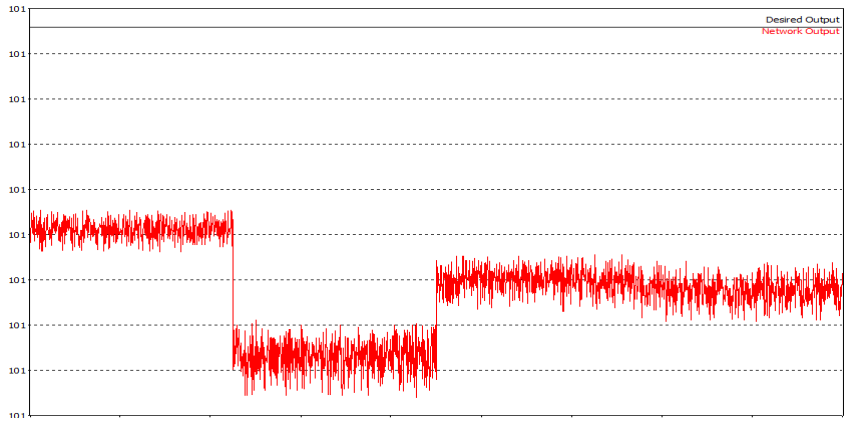
La figura D.2.3 muestra la gráfica generada por el valor escalar correspondiente a cada entrada del conjunto de entrenamiento, al utilizar una función SIGMOIDE de activación.

Tal cómo podemos apreciar en la figura D.2.3, si tenemos p clases linealmente no separables en el conjunto de entrenamiento, un solo perceptrón no tiene la capacidad de poder resolver el problema ya que para una misma clase se tienen diferentes rasgos y, por lo tanto, es necesario agregar un elemento extra para poder realizar una clasificación exitosa. Dicho elemento puede ser otra capa de neuronas o una regla de decisión a la salida de la neurona.

Por otra parte, la figura D.2.4 muestra la gráfica generada por el valor escalar correspondiente a cada entrada del conjunto de entrenamiento, al utilizar una función RAMPA de activación.

Tal cómo podemos apreciar en la figura D.2.4, la función rampa no genera rangos y por lo tanto tampoco es capaz de resolver el problema no lineal y, por lo tanto, al igual que en el caso de la sigmoide, requiere también la adición de un elemento extra.

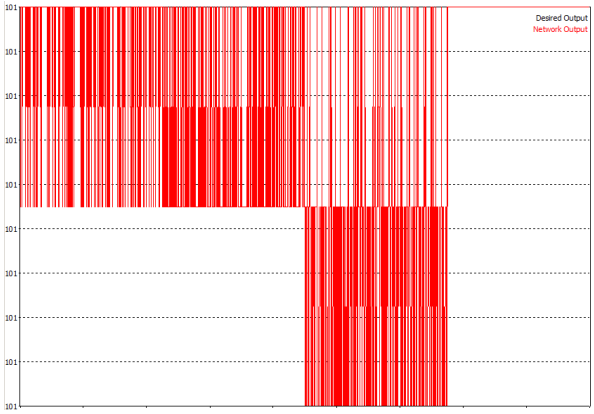
Valor escalar al utilizar la función sigmoide



Índice del conjunto de entrenamiento

Figura D.2.3. Salida escalar de cada entrada del conjunto de entrenamiento para 2 clases linealmente no separables y 2 rasgos.

Valor escalar al utilizar la función rampa



Índice del conjunto de entrenamiento

Figura D.2.4. Salida escalar de cada entrada del conjunto de entrenamiento para 2 clases linealmente no separables y 2 rasgos.

Referencias

[1] Wulfram Grestner (2001). “Spiking Neurons”. Pulsed Neural Networks. MIT Press.

[2] Xin Jin; Furber, S. B.; Woods, J. V. (2008). "Efficient modeling of spiking neural networks on a scalable chip multiprocessor". 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence).

[3] Eugene M. Izhikevich (2003). “Simple Model of Spiking Neurons”. IEEE Transactions on Neural Networks.

[4] Gerstner and Kistler (2002). Spiking Neuron Models. Single Neurons, Populations, Plasticity. Cambridge University Press.

[5] Roberto A. Vázquez (2010). “Pattern recognition using spiking neurons and firing rates”. Advances in Artificial Intelligence. IBERAMIA 2010.

[6] Tijn van der Zant (2009). “RoboCup@Home: Creating and Benchmarking Tomorrows Service Robots Applications”. RoboCup 2009.

[7] Una-May O’Reilly (2010). “Humanoids”. http://courses.csail.mit.edu/6.141/spring2010/pub/lectures/Lec19-Humanoids.pdf, 2012.

[8] Fulvo Mastrogiovanni (2012). “The need for a research agenda in intelligent robotics”. Intelligence Service Robotics.

[9] Heriberto Casarrubias, Juan Humberto Sossa, Ricardo Barrón (2007). “Generación de trayectorias para un robot móvil Khepera II usando técnicas de aprendizaje automático”. CIC-IPN.

[10] Sven Behnke (2008). “Humanoid Robots – From Fiction to Reality?”. Künstliche Intelligentz Heft.

[11] Raúl Rojas (1996). “Neural networks. A systematic introduction”. Springer Verlag.

[12] Bradley R Smith (1997). “Neural Network Enhancement of Closed-Loop Controllers for Ill-Modeled Systems with Unknown Nonlinearities”. Virginia Polytechnic Institute.

[13] Jilles Vreeken (2003). “Spiking Neural Networks: An Introduction”. Artificial Intelligence Laboratory. Intelligent Systems Group. Utrecht University.

[14] Avrim Blum (2010). “Machine learning theory”. Springer.

[15] B. Sofman, E. Lin, A. Bagnell, N. Vandapel (2006). “Improving robot navigation through self-supervised online learning”. Proceedings of Robotics: Science and Systems.

[16] S.B. Kotsiantis (2007). “Supervised machine learning: A review of classification techniques”. Department of Computer Science & Technology. University of Peloponnese.

[17] Ethem Alpaydin (2010). “Introduction to Machine Learning”. Second Edition. MIT Press. Cambridge, Massachusetts.

[18] R. Dillman, O. Rogalla, M. Ehrenmann, R. Zöllner (1999). “Learning Robot Behaviour and Skills Based on Human Demonstration and Advice: The Machine Learning Paradigm”. Proceedings of the Ninth International Symposium of Robotics Research.

[19] Nelson, M., Rinzel (1995). “Realistic Neural Models with the General Neural Simulation System”. Bower, J.M., Beeman, D. (Eds.).

[20] Jodi Forlizzi, Carl DiSalvo (2006). “Service Robots in the Domestic Environment: A Study of the Roomba Vacuum in the Home”. First ACM International Conference on Human Robot Interaction.

[21] H.Sahin, L, Guvenc (2007). “Household robotics: autonomous devices for vacuuming and lawn mowing”. IEEE Control Systems Magazine.

[22] Paul E. Rybski, Kevin Yoon, Jeremy Stolarz, Manuela Veloso (2006). CMAssist: A RoboCup@Home Team. Robotics Institute. Carnegie Mellon University.

[23] Keith Sullivan, Sean Luke, and Vittorio Amos Ziparo (2010). “Hierarchical Learning from Demonstration on Humanoid Robots”. Proceedings of Humanoid Robots Learning from Human Interaction Workshop.

[24] Nelson, M., Rinzel, (1995). Realistic Neural Models with the General Neural Simulation System. Bower, J.M., Beeman, D. (Eds.).

[25] G. Huerta, H. Sossa, E. Rubio (2009). “Localización y manipulación de objetos mediante un robot móvil Khepera II auxiliado de una cámara y un elemento prensil”. CIC-IPN.

[26] B. Argall , S. Chernova, M. Veloso, B. Browning (2008). “A survey of robot learning from demonstration”. Robotics and Autonomous Systems.

[27] M. N. Nicolescu, M. J. Mataric (2003).”Natural Methods for Robot Task Learning: Instructive Demonstrations, Generalization and Practice”. Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems.

[28] Gamez D, Fidjeland A K, Lazdins (2012). “iSpike: A spiking neural interface for the iCub robot”. Bioinspir. Biomim.

[29] C. P. Tung and A. C. Kak (1995). “Automatic learning of assembly tasks using a dataglove system”. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.

[30] E. W. Aboaf, S. M. Drucker, and C. G. Atkeson (1989). “Task-level robot learning: Juggling a tennis ball more accurately”.IEEE International conference on Robotics and Automation.

[31] P. Domingos (2012). A few useful things to know about machine learning. Communications of the ACM.

[32] Mitchell, T. (1997).Machine Learning, McGraw Hill.

[33] Breazeal, C and Scassellati, (2002). “Robots that imitate humans”. Trends in Cognitive Science.

[34] Wulfram Gerstner (2002). “Integrate-and-fire neurons and networks”. The handbook of brain theory and neural networks. MIT Press.

[35] Larry Yaeger (2010). “Spiking Neuron Models”. InformaticsIndiana University.

[36] A. Abate, F.D. Dalmoro (2007) .”Support network machine versus artificial network. Who is the winner?”. Kidney Int.

[37] Carlos Gerghenson (2003). “Artificial Neuronal Networks for Beginners”. United Kingdom.

[38] Linda G. Shapiro and George C. Stockman (2001). “Computer Vision”. Prentice Hall.

[39] David A. Forsyth and Jean Ponce (2003). “Computer Vision, A Modern Approach2. Prentice Hall.

[40] Dana H. Ballard and Christopher M. Brown (1982). “Computer Vision”. Prentice Hall.

[41] González and Woods (2008). “Digital Image Processing”. Prentice Hall.

[42] Humberto Sossa (2006). “Rasgos descriptores para el reconocimiento de objetos”. Insituto Politécnico Nacional. Ciencia de la computación.

[43] K.N. Plataniotis, A.N. Venetsanopoulos (2000). “Color Image Processing and Applications”. Springer-Verlag.

[44] W. K. Pratt (2007), “Digital Image Processing”. John Wiley & Sons, Inc.

[45] M. Petrou, P. Bosdogianni (2004).” Image Processing the Fundamentals”. Wiley.

[46] M.K. Hu (1962). “Visual pattern recognition by moment invariants”. IRE Transactions on information theory.

[47] S. Dalibar, A.Nakhaei, F. Lamiraux, (2010). “Manipulation of documented objets by a walking humanoid robot”, IEEE-RAS International Conference Humanoid robots.

[48] S. Tolu, E. Ros, R. Agís (2009). “Bio-inspired control model for object manipulation by humanoid robots,”. International Work Conference on Artificial Neural Networks.

[49] P. Nasrollahi, S. Jafari, M. Ebrahimi (2012). “Action Classification of Humanoid Soccer Robots Using Machine Learning”. Artificial Intelligence and Signal Processing.

[50] V. Tikhanoff, A. Cangelosi, G. Metta (2011). “Integration of Speech and Action in Humanoid Robots: iCub Simulation Experiments”. IEEE Transactions on Autonomous Mental Development.

[51] A. Maughlbay, E. Cervera, P. Martinet (2013). “Model Based Visual Servoing Taks with an Autonomus Humanoid Robot”. Frontiers of Intelligent Autonomus Systems. Springer.

[52] Y. Pang, Q. Huang, D Jia (2008). “Object Manipulation of a Humanoid Robot Based on Visual servoing”. International Conference on Intelligent Robots and Systems San Diego.

[53] M. Kojima, K. Okada M. Inaba (2009). “Manipulation and Recognition of Objects Incorporating Joints by a Humanoid Robot for Daily Assistive Tasks”. International Conference on Intelligent Robots and Systems.

[54] Fabian Chersi (2012). “Learning Through Imitation: a Biological Approach to Robotics”. IEEE Transactions on Autonomous Mental Development.

[55] K. Harada, S. Kaijita, H. Saito (2005). “A Humanoid Robot Carrying a Heavy Object”. . International Conference on Robotics and Automation.

[56] S. Oswald, A. Hornung, M. Bennewitz (2010). “Learning Reliable and Efficent Navigation with a Humanoid Robot”. IEEE International Conference on Robotics and Automation.

[57] T. Leugger and S. Nolfi (2012). “Action Development and Integration in a Humanoid iCub Robot”. Biomimetic and Biohybrid Systems. Springer.

[58] H. Masuta and N. Kubota (2010). “Perceptual system using spiking neural netork for an intelligent robot”. International conference on Systems Man and Cybernetics.

[59] D. Katic (2004). “Intelligent control of humanoid robots using neural networks”. Neural Network Applications in Electrical Engeneering.

[60] M. Capuozzo (2011). “A compact evolutionary algorithm for integer spiking neural network robot controllers”. Proceedings of IEEE Southeaston.

[61] Paul Wright (2012). “Learning transmission delays in spiking neural networks: A novel approach to sequence learning based on spike delay variance”. International Joint conference on Neural Networks.

[62] M. Stolca, G.A. Calagniu and I. Sarkany (2010). “A method proposed for training an artificial neural network used for industrial robot programming by demonstration”. International Conference on Optimization of electrical and Electronic Equipment.

[63] A. Bougnais and P. Murray (2010). “Training a spiking neural network to control a 4-D-oF robotic arm based on Spike Timing-Dependent Plasticity”. International Joint Conference on Neural Networks.