“Navegación autónoma de un robot con técnicas de localización y ruteo”

Por:

Mariana Natalia Ibarra Bonilla

Tesis sometida como requisito parcial para obtener el grado de

MAESTRÍA EN CIENCIAS EN LA ESPECIALIDAD DE ELECTRÓNICA.

En el

Instituto Nacional de Astrofísica, Óptica y Electrónica.
Diciembre 2009
Tonantzintla, Puebla

Supervisada por:

Dr. Juan Manuel Ramírez Cortés
Investigador titular del INAOE

Resumen

En este trabajo se presenta el diseño e implementación de un sistema de visión por computadora y control difuso orientado a la navegación autonoma de un robot móvil Lego NXT con capacidades de localización y mapeo simultáneo en un ambiente controlado. Así mismo, se describe el algoritmo de navegación implementado para el desempeño de diversas tareas, tales como la búsqueda de la ruta más cercana en base al algoritmo de Dijkstra, la detección y evasión de los obstáculos presentes en su entorno y la alineación en el desplazamiento de la ruta basada en visión y lógica difusa. La arquitectura móvil se encuentra equipada con una cámara web y un sensor de ultrasonido para la percepción del ambiente y de los obstáculos presentes en el mismo. El sistema ha sido desarrollado en Matlab utilizando una interfaz gráfica para simular la posición y movimiento del móvil, así como las cajas de herramientas de lógica difusa y de adquisición y tratamiento de imágenes.

Abstract

This work presents the design and implementation of a computer vision system and fuzzy control oriented autonomous navigation of a mobile robot Lego NXT with capabilities simultaneous localization and mapping in a controlled environment. This work also describes the implemented navigation algorithm to execute different tasks such as finding the shortest path based on Dijkstra's algorithm, the detection and avoidance of obstacles in their environment and and alignment in the displacement of the path based on vision and fuzzy logic. The mobile architecture is equipped with a webcam and an ultrasonic sensor for the perception of the environment and the obstacles. The system has been developed in Matlab using a graphical interface to simulate the position and movement of the mobile and the fuzzy logic and image acquisition and processing toolboxes.

Agradecimientos

Agradezco principalmente a Dios por estar presente en mi vida y por darme la fortaleza para continuar adelante.

A mis padres, por el apoyo, amor y comprensión que me han brindado día con día y por creer en mí por sobre todas las cosas.

A mis hermanos, por contar con ellos en todo momento.

A mi novio, por apoyarme incondicionalmente y darme su comprensión.

A mi asesor de tesis, Dr. Juan Manuel Ramírez Cortés por el tiempo que me dedicó para la realización de esta tesis. Muchas gracias por su amabilidad y confianza.

A mis maestros y sinodales: Dr. Jorge Martínez Carballido, Dr. Rogerio Enríquez Caldera y M.C. Jorge Pedraza Chávez.

Al INAOE y CONACYT, por el apoyo y la beca otorgada para la realización de mis estudios de Maestría.

Dedicatoria

A Lucia y José Luis

Con amor, respeto y admiración.

Por todos sus sacrificios, esfuerzos y dedicación y

por ser los mejores padres que pude tener.

A Fernando

Por haber confiado y creído siempre en mí.

Índice General

Capítulo 1

Introducción

La necesidad de incrementar la autonomía en las aplicaciones robóticas ha motivado la creación y desarrollo de robots móviles. El propósito es limitar en todo lo posible la intervención humana. Para ello, el robot debe poseer la suficiente inteligencia para reaccionar y tomar las decisiones basándose en observaciones de su entorno, aún cuando éste sea completamente desconocido. La autonomía de un robot móvil se basa principalmente en el sistema de navegación autónoma. En éstos sistemas incluyen diversas tareas como: planificación, percepción y control.

La planificación en los robots móviles, consiste generalmente en establecer la misión, la ruta y la evasión de obstáculos. Por ejemplo, en un robot para interiores, la misión podría consistir a qué habitación desplazarse, mientras que la ruta establece el camino desde la posición inicial a una posición determinada dentro de la habitación. Sin embargo, ésta última puede ser corregida debido a la presencia de objetos no esperados dentro del entorno.

El sistema de percepción tiene un triple objetivo: permitir una navegación segura mediante la detección y localización de los obstáculos y situaciones peligrosas en general, modelar el entorno construyendo un mapa o representación de este último, y conocer la posición precisa del vehículo. Esta etapa puede requerir el uso de diversos sensores, tales como, ultrasonido, sistemas láser, sonar, o cámaras de video.

Una vez obtenido el mapa del entorno, el objetivo se centra en su representación topológica y en los algoritmos de navegación pertinentes a la aplicación deseada, como por ejemplo el algoritmo de Dijkstra para la determinación de la ruta más cercana [1]. Por otra parte, la aplicación del control difuso en los sistemas de navegación permite tomar decisiones en forma de razonamiento aproximado al ser humano de acuerdo a la información contenida en una base de conocimientos.

De igual forma, diversos sistemas basados en localización visual orientados a la navegación automática han sido reportados con buenos resultados [2, 3]. Las técnicas de procesamiento de imágenes adquiridas a través de una o varias cámaras, utilizadas como elemento de sensado para cerrar el lazo de control presentan alternativas muy interesantes en sistemas de navegación [4, 5].

1.1 Planteamiento del problema

Como ya se mencionó, un sistema de navegación autónoma consiste básicamente de un vehículo que no requiere de un operador para navegar y cumplir con sus diversas tareas. Actualmente, existen un gran número de aplicaciones donde los vehículos autónomos son empleados con éxito: exploración espacial, manejo de explosivos, agricultura, transporte automático en áreas urbanas, sistemas de seguridad, así como exploración en sitios de riesgo para el ser humano [6, 7, 8].

Estos sistemas deben contar con un control, el cual permita tomar las decisiones adecuadas para su desplazamiento automático y las demás tareas para las que haya sido concebido; por ejemplo, la evitación de obstáculos para alcanzar un objetivo sin perder su ruta. Para alcanzar sus objetivos, el robot debe contar con una serie de sensores que le permita percibir el entorno que lo rodea.

Con el uso de los sensores, es necesario tener en cuenta las características de precisión, rango, e inmunidad a la variación de condiciones del entorno. Por ejemplo, las cámaras de video tienen la ventaja de su amplia difusión y precio. Las desventajas son los requerimientos computacionales, la sensibilidad a las condiciones de iluminación y los problemas de calibración [9].

Los sensores de ultrasonido son económicos y simples para la navegación. Sin embargo, la resolución lateral es mala, por lo que es necesario distribuir más de uno sobre el robot móvil [9].

1.1.1 Motivación y Justificación

De acuerdo a lo antes descrito, se puede concluir que la complejidad de la navegación de un robot móvil depende principalmente del control. Partiendo del interés de contribuir con un sistema de navegación simple y eficiente; se propone la implementación de un algoritmo capaz de hacer que una arquitectura móvil cumpla con el propósito de recorrer la ruta más corta libre de colisiones, desde un inicio hasta una meta; y que a su vez, sirva como una plataforma didáctica que ayude a fomentar el interés en el estudio de las aplicaciones de la robótica móvil.

Existen diversos algoritmos de enrutamiento por la ruta más corta; sin embargo, el Algoritmo de Dijkstra ha brindado buenos resultados en las diferentes aplicaciones existentes del mismo [10, 11]. Por consecuencia, se ha convertido en uno de los más

populares en la ciencia computacional, aumentando así la importancia de su aprendizaje. La facilidad de su aplicación y el bajo nivel de procesamiento del mismo ofrecen una buena alternativa como algoritmo de enrutamiento en un sistema de navegación.

Se incorpora en el sistema el uso de un sensor de ultrasonido en adición con una cámara de video, para compensar las limitaciones que presentan por separado cada uno de ellos. Por lo tanto, se decidió utilizar la arquitectura móvil Lego NXT por incluir el sensor ultrasónico y la conexión inalámbrica Bluetooth.

La corrección de la inevitable acumulación del error está presente en los sistemas de navegación. La lógica difusa y el procesamiento de imágenes han demostrado resultados favorables en aplicaciones de detección y seguimiento de algún objeto [5], por lo que resultan ser una buena opción para realizar los ajustes en la localización del robot.

1.2 Objetivos

El objetivo general de este trabajo es el diseño e implementación de un sistema de visión por computadora y control difuso orientado a la navegación autónoma con capacidades de localización y mapeo simultáneo de un robot móvil LEGO NXT.

Como objetivos particulares se pretende detectar los obstáculos presentes en el entorno. Incorporar la búsqueda de la ruta más cercana y libre de colisiones en base al algoritmo de Dijkstra. Realizar ajustes sobre la localización del móvil dentro de la representación del entorno por medio de operaciones de control difuso.

El objetivo final de este proyecto es que el sistema completo sirva como apoyo didáctico que permita posteriormente experimentar con diversas técnicas de posicionamiento, generación simultánea y navegación automática, contando para tal efecto con diversos tipos de sensores.

1.3 Limitaciones de la Solución

Este trabajo pretende ser empleado en un escenario pequeño mapeado hacia una rejilla para su posterior representación en forma de nodos. Por lo que solo se implementará en ambientes controlados de dos dimensiones.

El sistema únicamente propone detectar obstáculos y no será capaz de reconocer la estructura geométrica del mismo. Así mismo, el uso del sensor de ultrasonido y la cámara de video serán empleados independientemente para llevar a cabo dicho objetivo.

1.4 Organización de la Tesis

En esta sección se proporciona una perspectiva general de la organización del documento, y se presenta una visión general de lo que se aborda en cada uno de los capítulos de la tesis.

El capítulo 2 presenta una recopilación de la información referente a la navegación autónoma de robots móviles. Se incorpora una descripción de los sistemas SLAM, los algoritmos de enrutamiento y los fundamentos básicos de la lógica difusa.

En el capítulo 3, se presenta la descripción por etapas del diseño e implementación del sistema de navegación.

En el capítulo 4, se muestran los resultados obtenidos de los diferentes experimentos realizados para validar el funcionamiento del sistema de navegación.

Por último, en el capítulo 5 se presentan las conclusiones obtenidas en la implementación de este trabajo, así como también el trabajo futuro que puede desarrollarse para extender y mejorar el sistema.

Capítulo 2

Marco Teórico

En este capítulo se hace una introducción a los temas relacionados con el problema a desarrollar en este trabajo. Inicia con una introducción a los sistemas SLAM, los cuales son utilizados en los sistemas de localización y mapeo de robots móviles. Posteriormente, se hace una revisión de los diferentes algoritmos de enrutamiento por la ruta más corta, los cuales son esenciales para describir formalmente al algoritmo de Dijkstra involucrado en este proyecto. Por último, se describen los conceptos básicos de la lógica difusa y como ésta se utiliza en las aplicaciones de navegación robótica.

2.1 Sistemas SLAM

2.1.1 Introducción a los sistemas SLAM

El proceso de posicionamiento y la generación simultánea de un mapa del entorno es formalmente conocido como SLAM por las siglas en inglés (*Simultaneous localization and mapping*). Localización y mapeo simultáneo, SLAM, es una técnica

usada en robots y vehículos autónomos para construir un mapa dentro de un ambiente desconocido, aún cuando controlado en muchos casos, manteniendo registro de su posición en todo momento.

A la fecha se han estudiado aplicaciones SLAM en una gran variedad de dominios interiores y exteriores, incluyendo lugares muy apartados [12], muy peligrosos [6] o simplemente en lugares donde el acceso humano resulta muy costoso.

Si los robots son operados de manera autónoma en ambientes extremos bajo el mar, bajo la tierra y sobre superficies de otros planetas, ellos deben ser capaces de construir mapas y navegar correctamente de acuerdo a esos mapas [13]. Para adquirir el modelo del ambiente de un robot, un sistema SLAM puede requerir el uso de diversos sensores, tales como, ultrasonido, sistemas láser, sonar, o cámaras de video.

(a) Bajo el mar (b) Bajo tierra (c) Otros planetas

Fig. 2. 1 Diferentes ambientes para aplicaciones SLAM

Los sensores de movimiento propios del robot obtienen información importante para la construcción del mapa; sin embargo estas lecturas están sujetas a errores, principalmente por errores en el movimiento del robot y el ruido. En general, los errores en la ruta del robot originan errores en el mapa. Como resultado, para obtener un mapa verdadero del ambiente se requiere una constante localización del robot.

La Figura 2.2 muestra un escaneo láser obtenido por un robot móvil moviéndose a través de un ambiente interior típico [13]. El robot genera la estimación de su

posición usando odómetros adheridos a cada una de sus llantas. En la Figura 2.2-a, se muestra el mapa generado sin el trabajo de la localización, únicamente utilizando los datos obtenidos del sensor. Como puede verse el error es acumulativo, mientras más se desplaza el robot, el error registrado es mayor. Por consecuencia, el mapa llega a ser cada vez más inexacto. En la Figura 2.2-b se aprecia un mapa trazado de acuerdo con la ruta que el robot reconstruyó con un algoritmo SLAM.

(a) Mapa construido sin la corrección en la ruta del robot

(b) Mapa construido usando SLAM

Fig. 2. 2 Correlación entre el error en la ruta del robot y el error en el mapa

2.1.2 El problema del SLAM

Las investigaciones y trabajos científicos se han enfocado en resolver el problema de cómo generar un mapa del entorno tomando en cuenta los errores acumulados por la lectura de los sensores [14, 15]. Para actualizar el mapa visible por los sensores del robot, se debe conocer la posición exacta del móvil. Sin embargo; para localizarse, el robot requiere un mapa previo debido a que no puede confiar en la información obtenida de los sensores. El mapa previo no existe, porque se está haciendo en ese momento. En general, no puede hacerse un mapa sin localización y no se puede localizar un robot sin un mapa [16]. Como resultado, la localización y el mapeo deben ser estimados simultáneamente.

En la actualidad han sido abordados diversos enfoques para la solución del problema de SLAM, especialmente desde el punto de vista probabilístico con un enfoque Bayesiano, con soluciones que involucran técnicas del tipo filtrado Kalman, aplicado generalmente a ambientes reconstruidos con sensores láser [12, 17**]**.

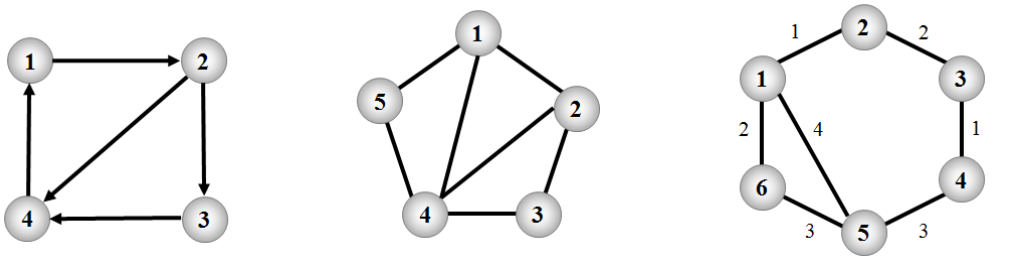
2.2 Algoritmos de enrutamiento por la ruta más corta

Una vez obtenido el mapa del entorno, el objetivo de un sistema de navegación autónoma se centra en su representación topológica y en los algoritmos de enrutamiento pertinentes a la aplicación deseada. En este trabajo se busca conseguir que el robot móvil sea capaz de recorrer la ruta más corta libre de colisiones. Por lo tanto, la investigación se centra en los principales algoritmos que resuelven este problema.

2.2.1 Definiciones básicas

Definición 2.1. *Grafo*. Un grafo G es la combinación de un conjunto de vértices o nodos V y un conjunto de aristas A que conectan a los vértices. Se representa como $G = [V, A]$, donde el número de vértices es $|V| = n$, y el número de aristas es $|A| = m$. Un grafo conectado comprende al menos un camino o ruta desde algún vértice v a cualquier otro vértice u, [18]. La figura 2.3-a muestra un grafo $G = [V, A]$, donde $V = [1, 2, 3, 4]$ y $A = [(1,2), (2,3), (2,4), (3,4), (4,1)]$.

Si el grafo tiene atribuido un peso en sus aristas mediante la función $w: A \rightarrow R+$, se denomina grafo ponderado. La función que describe la longitud de la arista es $w(v_1, v_2)$. Si dos vértices v_1 y v_2 no están conectados por una arista $a \in A$, se define $w(v_1, v_2)$ como ∞ [18].



(a) Grafo dirigido (b) Grafo no dirigido (c) Grafo ponderado

Fig. 2. 3 Ejemplos de dos diferentes tipos de grafos

Una ruta de longitud *k* desde un vértice *v* a un vértice *v'* en un grafo $G = [V, A]$, es una secuencia $[v_0, v_1, v_2, \ldots, v_k]$ de vértices, tal como $v = v_0$, $v' = v_k$ y aristas $(v_{i-1}, v_i) \in A$ para $i = 1,2,\ldots, k$. La longitud de una ruta es la suma de los pesos de las aristas contenidos en ella. Una trayectoria contiene los vértices $v_0, v_1, \ldots v_k$ y las aristas $(v_0, v_1), (v_1, v_2), \ldots (v_0, v_k)$. Si existe una ruta *p* desde *v* hasta *v'*, se dice que *v'* es *accesible* o *alcanzable* desde *v* a través de *p*, lo cual se denota $v \xrightarrow{p} v'$ [19]. En la Figura 2.3-b, la trayectoria (1, 2, 3, 4, 5) contiene las aristas (1, 2), (2, 3), (3, 4) y (4, 5) y su longitud es 7.

Definición 2.2. Problema de la ruta más corta. El problema de la ruta más corta en un grafo ponderado consiste en encontrar el camino más corto entre un nodo origen y un nodo destino. Dado un grafo $G = [V, A]$ con funciones de peso $w: A \rightarrow R+$, el peso de la ruta $p = (v_0, v_1, v_2, \ldots, v_k)$ es la suma de los pesos de sus aristas.

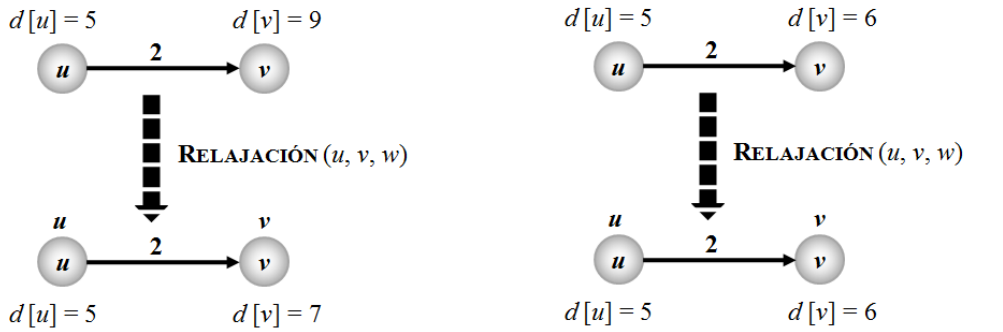
$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \tag{2.1}$$

La ruta más corta desde *u* hasta *v* se define por:

$$\delta(u, v) \begin{cases} \min\{w(p) : u \xrightarrow{p} v & \text{si hay una ruta desde } u \text{ hasta } v \\ \infty & \text{en otro caso} \end{cases} \tag{2.2}$$

Entonces, la ruta más corta desde el vértice *u* al vértice *v* está definida como cualquier ruta *p* cuyo peso sea $w(p) = \delta(u,v)$ [19].

Definición 2.3. Relajación. Dado un grafo $G = [V, A]$, por cada vértice $v \in V$ se mantiene un atributo $d[v]$, el cual se estima como la ruta más corta desde un origen *s* a *v*. De igual forma, se mantiene un predecesor $\pi[v]$, el cual puede ser cualquier otro vértice unido directamente a *v* [19]. En el caso del vértice origen no hay predecesor. La Figura 2.4 muestra dos ejemplos de relajamiento sobre una arista, en uno de ellos la ruta más corta estimada decrece y en el otro caso no hay cambios. La Tabla 2.1 presenta en pseudocódigo el algoritmo de relajación ejecutado sobre una arista (u, v).



(a) $d[v] > d[u] + w(u, v)$ antes de relajar por lo que $d[v]$ decrece

(b) $d[v] \leq d[u] + w(u, v)$ antes de la relajar por lo que $d[v]$ no cambia

Fig. 2. 4 Relajación sobre la arista (u, v) con peso $w(u, v) = 2$

Tabla 2. 1 Pseudocódigo del Algoritmo de Relajación

Algoritmo 2.1 Relajación

```
INICIALIZAR- GRAFO – ORIGEN (G, s)
1:      for cada vértice v ∈ V[G]
2:              do d[v] = ∞
3:                  π[v] = nulo
4:      d[s] = 0
5:      end for
RELAJACIÓN (u, v, w)
6:      if d[v] > d[u] + w (u, v)
7:              then  d[v] = d[u] + w (u, v)
8:                      π[v] = u
9:      end if
```

2.2.2 Diferentes algoritmos de cálculo de la ruta más corta

Desde la década de los cincuenta, los matemáticos han concentrado sus esfuerzos por desarrollar algoritmos que puedan determinar la ruta más corta entre dos puntos: un origen y un destino pertenecientes a una red o grafo [19]. Entre los más importantes se encuentran:

- El algoritmo de Dijkstra.
- El algoritmo de Bellman-Ford.
- Algoritmo de Floyd-Warshall.
- Algoritmo de Johnson.

El **Algoritmo de Dijkstra** resuelve el problema de la ruta más corta, desde un vértice origen al resto de los vértices contenidos en un grafo ponderado. La idea principal en este algoritmo consiste en examinar las rutas que parten del vértice origen hacia todos los demás vértices. Cuando se obtiene el camino más corto desde el origen al resto de los vértices que componen el grafo, el algoritmo finaliza. La descripción de este algoritmo se detallará en la sección 2.3.

El **Algoritmo de Bellman-Ford** al igual que el algoritmo de Dijkstra, genera la ruta más corta desde un nodo o vértice origen en un grafo ponderado al resto de los vértices del mismo. A diferencia de este último, el algoritmo de Bellman-Ford soluciona el problema de un modo más general, ya que permite valores negativos en los arcos [20].

Dado un grafo ponderado y dirigido $G = [V, A]$, con un origen s y la función de peso w: $A \rightarrow \mathrm{R}$; el algoritmo Bellman-Ford retorna un valor booleano indicando si encuentra un circuito lazo de peso negativo. Si dicho lazo existe, el algoritmo indica que no hay solución. En caso contrario, el algoritmo calcula y devuelve el camino más corto con su peso [19].

El algoritmo usa la técnica de relajación, decreciendo progresivamente la estimación de la ruta más corta $d[v]$ desde el origen s en cada vértice $v \in V$ hasta conseguir la ruta más corta real $\delta(s, v)$. El algoritmo retorna VERDADERO si y solo si la ruta más corta obtenida no es de peso negativo. La Tabla 2.2 presenta en pseudocódigo el algoritmo de Bellman-Ford.

Tabla 2. 2 Pseudocódigo del Algoritmo de Bellman-Ford

Algoritmo 2.2 Bellman-Ford

```
BELLMAN-FORD (G, w, s)
1:  INICIALIZAR - GRAFO-ORIGEN (G, s)
2:  for cada vértice v ∈ V [G] do
3:        d[v] = ∞
4:        π[v] = nulo
5:   d[s] = 0
6:  end for
7:  for i = 1 to V [G] - 1
8:  do for cada arista (u, v) ∈ A[G] do RELAJACIÓN
9:              if   d[v] > d[u] + w (u, v)   then
10:                  d[v] = d[u] + w (u, v)
11:                  π[v] = u
12:             end if
13:  for cada arista (u, v) examinar si hay lazo de peso negativo do
14:       if d[v] > d[u] + w (u, v)
15:            then return FALSO el algoritmo no converge
16:       else return VERDADERO
17:       end if
18:  end for
```

El algoritmo de **Floyd-Warshall** encuentra el camino más corto entre todos los pares de vértices en un grafo dirigido $G = [V, A]$ en una única ejecución. El algoritmo considera los vértices "intermedios" de una ruta más corta. El vértice *intermedio* de una ruta sencilla $p = (v_1, v_2, \ldots, v_l)$ es cualquier vértice de p excepto v_1 o v_l, esto es, cualquier vértice en el conjunto $(v_2, v_3, \ldots, v_{l-1})$.

El algoritmo Floyd-Warshall está basado en la siguiente observación: suponiendo que los vértices de G son $V = [1,2,\ldots, n]$ y considerando un subconjunto $[1,2,\ldots, k]$ de vértices.

Para cualquier par de vértices $i, j \in V$, se deben tomar en cuenta todas las rutas desde i a j cuyos vértices intermedios se encuentren dentro del subconjunto [1,2,…, k]; p se considera la ruta más corta entre todas ellas. El algoritmo Floyd-Warshall explota la relación entre p y sus vértices intermedios. Esta relación depende si k es o no es un vértice intermedio de la ruta p:

- Si k no es un vértice intermedio de la ruta p, entonces todos los vértices intermedios de la ruta p están en el conjunto [1,2,…, k–1]. De esta manera, el subconjunto [1,2,…, k–1] es la ruta más corta de i hasta j del conjunto [1,2,…, k].
- Si k es un vértice intermedio de la ruta p, entonces se descompone la ruta p en $i \xrightarrow{p1} k \xrightarrow{p2} j$, tal como se muestra en la Figura 2.5. En general, p_1 es la ruta más corta desde i a k con vértices intermedios en el conjunto [1,2,…, k]. Debido a que el vértice k no es un vértice intermedio de la ruta p_1, observamos que p_1 es la ruta más corta desde i hasta k con vértices intermedios en [1,2,…, k–1]. Similarmente, p_2 es la ruta más corta desde k a j con vértices intermedios en el subconjunto [1,2,…, k–1] [19].

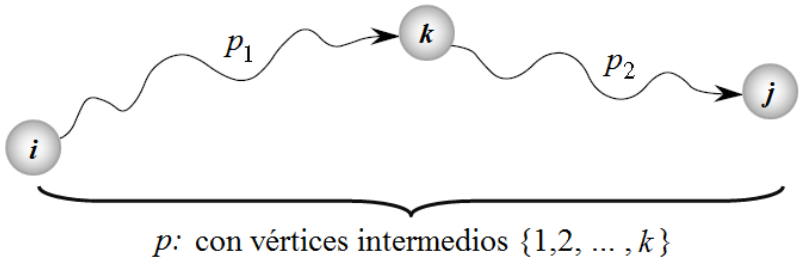


Fig. 2. 5 Descomposición de la ruta más corta p en p1 y p2

Basado en las observaciones anteriores, se define la ruta más corta en base a la formula recursiva 2.3. La Tabla 2.3 detalla el pseudocódigo del algoritmo de Floyd-Warshall.

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \textbf{if } k = 0, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \textbf{if } k \geq 1. \end{cases} \tag{2.3}$$

Tabla 2. 3 Pseudocódigo del Algoritmo de Floyd-Warshall

Algoritmo 2.3. Floyd-Warshall

```
FLOYD-WARSHALL (G, i, j, k)
Requiere: G grafo, i nodo inicio, j nodo final, k vértice intermedio
1:  for  i = 1 to n do
2:       for  j = 1 to n do
3:            d[i, j] = w (i, j)
4:       end for
5:  end for
4:  for k = 1 to n do
5:       for  i = 1 to n do
8:            for  j = 1 to n do
9:                 if   d[i, k] + d[k, j] < d[i, j]  then
10:                      d[i, j] = d[i, k] + d[k, j]
11:                end if
12:           end for
13:      end for
14: end for
```

El **Algoritmo de Johnson** soluciona el problema de la ruta más corta en un grafo dirigido $G = [V, A]$ haciendo uso de los algoritmos de Dijkstra y de Bellman-Ford. El algoritmo permite el uso de aristas con peso negativo, por lo que utiliza el algoritmo de Bellman-Ford para eliminar las aristas de peso negativo. Posteriormente se ejecuta el algoritmo de Dijkstra sobre el grafo transformado.

Los algoritmos de Floyd-Warshall y de Johnson son más complejos en comparación con los algoritmos de Dijkstra y Bellman-Ford. Por su parte, el algoritmo de Dijkstra resuelve el problema de la ruta más corta en menor tiempo que el de Bellman-Ford, ya que éste no acepta aristas con pesos negativos.

En este trabajo no se busca obtener un grafo con aristas de peso negativo; por lo tanto, el algoritmo de Dijkstra es el mejor candidato a implementar gracias a la rapidez en su ejecución. La siguiente sección detalla el funcionamiento de este algoritmo.

2.3 Algoritmo de Dijkstra

El algoritmo de Dijkstra sirve para encontrar la ruta más corta desde un nodo origen a todos los demás nodos contenidos en el grafo ponderado $G = [V, A]$. Fue diseñado en 1959 por el holandés Edsger Wybe Dijkstra por el cual lleva su nombre. Este algoritmo considera $w(u, v) \geq 0$ por cada arista $(u,v) \in A$.

El algoritmo funciona de la siguiente manera: iniciando desde un vértice origen s, todos los nodos se etiquetan con su distancia al vértice origen $d[v]$. El algoritmo utiliza la técnica de relajación; inicialmente no se conocen las rutas, pero a medida que avanza el algoritmo y se encuentran las demás, las etiquetas pueden cambiar, reflejando mejores rutas. En un inicio, todas las etiquetas son temporales. Una vez que se descubre que una etiqueta representa la ruta más corta posible del origen a ese nodo, se vuelve permanente y no cambia más [18].

Tabla 2. 4 Pseudocódigo del Algoritmo de Dijkstra

Algoritmo 2.4. Dijkstra

DIJKSTRA (*G, w, s*)
Entrada: requiere un vértice origen s y una matriz de distancias d donde $d[i, j]$ contiene la distancia desde el nodo i a j; $d[i, j] = \infty$ cuando no hay trayectoria directa desde i a j.
Salida: Regresa la distancia $d[v]$ desde s a cualquier otro vértice $v \in V$.

```
1:  for cada vértice v ∈ V do
2:      d[v] = ∞
3:  end for
4:  d[s] = 0
5:  S = 0
4:  while existan nodos que no han sido etiquetados permanentemente, V \ S ≠ 0 do
5:      u = argmin {d[v]}
8:      S = S ∪ u
9:      for cada arista (u, v) do RELAJACIÓN
10:         if d[v] > d[u] + w (u, v) then
11:             d[v] = d[u] + w (u, v)
12:         end if
13:     end for
14: end while
```

El pseudocódigo del algoritmo de Dijkstra se describe en la Tabla 2.4. Los vértices etiquetados permanentemente se almacenan en *S*. La etiqueta permanente del nodo origen contiene $d[s] = 0$, ya que $dist\ [s, s] = 0$. Por cada vértice alcanzado *v*, una etiqueta temporal $d[v]$ contiene la distancia desde el origen; durante el primer ciclo esta es simplemente la longitud de la arista desde *s* a *v*. Se asigna al vértice *u* la etiqueta temporal con la distancia más corta $d[v]$. Si no existe una ruta más corta para llegar al vértice *v*, su etiqueta se vuelve permanente. Para determinar cuál es la ruta más corta, el algoritmo utiliza relajación.

Después de que el algoritmo de Dijkstra finaliza, todos los vértices contenidos en el grafo están etiquetados con la distancia más corta que se necesita recorrer a partir del origen *s*. Esas cantidades son usadas para obtener el recorrido desde el vértice final (*t*) al vértice origen. Por ejemplo, el vértice *x* que satisface: $d(x) = d(t) - w(x, t)$, debe ser el vértice previo con la ruta más corta hacia *t* [18]. La ejecución del Algoritmo de Dijkstra se ilustra en la Figura 2.6 [19].

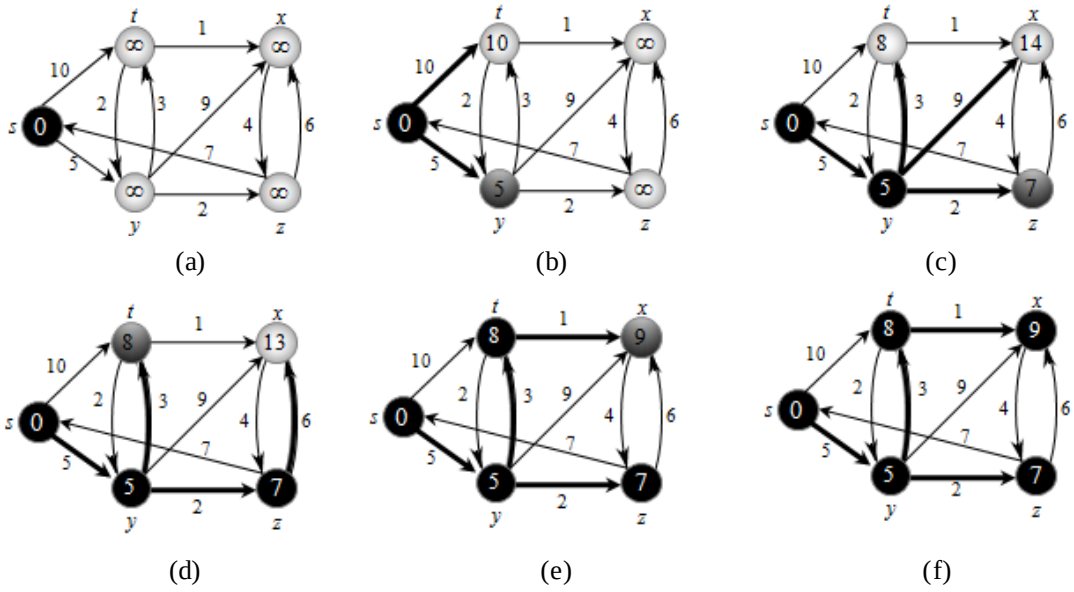


Fig. 2. 6 Ejecución del Algoritmo de Dijkstra

De acuerdo con la Figura 2.6, la distancia más corta estimada está indicada dentro de los vértices, y las aristas en negritas indican los valores predecesores. Los vértices en negro están en el conjunto *S*. La Figura 2.6-a muestra la situación antes de la primera iteración del ciclo **while** de las líneas 4-14 de la Tabla 2.4. Los vértices sombreados tienen el valor *d* mínimo, el cual es elegido como el vértice *u* en la línea 5. Las Figuras 2.6-b y 2.6-f muestran la situación después de cada iteración consecutiva del ciclo while. El vértice sombreado en cada parte es elegido como el vértice *u* en la línea 5 de la siguiente iteración. Los valores finales son mostrados en la Figura 2.6-f.

2.3.1 Algoritmo de Dijkstra etiquetado

En aplicaciones donde es importante conservar el nombre de cada nodo ó vértice, existe una manera distinta de ejecutar el algoritmo de Dijkstra. Esta consiste en asignar una etiqueta $[d(v), \pi(v)]$ a cada vértice $v \in V$ la cual contenga la distancia $d[v]$ al nodo de origen a través de la mejor ruta conocida y su nodo de procedencia $\pi[v]$, con la finalidad de poder reconstruir más tarde la ruta final. Este modo de ejecutar el algoritmo se denominará en este trabajo *Algoritmo de Dijkstra etiquetado*.

La ejecución del algoritmo etiquetado es similar a la descrita por la Tabla 2.4. Para ilustrar su funcionamiento, considere el grafo ponderado no dirigido de la Figura 2.7, donde las ponderaciones representan las distancias. Se inicia marcando como permanente el nodo ó vértice *A*, indicado con el relleno negro en la Figura 2.7-a. La etiqueta de este vértice es [0,–]; que indica que éste nodo no tiene predecesor. Se analizan, uno por uno, los vértices conectados directamente al permanente, etiquetando tentativamente la distancia acumulada desde A y el vértice de procedencia en cada uno.

De acuerdo al renglón 5 de la Tabla 2.4, el siguiente nodo permanente será el que contenga una menor distancia acumulada. En ésta primera iteración, los vértices *B* y *C* contienen la misma distancia acumulada; en este caso, se elige al azar el siguiente nodo permanente. El algoritmo repite el mismo procedimiento, ahora comenzando desde el vértice *B* como se muestra en la Figura 2.7-b.

En las Figuras 2.7-c y 2.7-e se ilustra la técnica de relajación. Por ejemplo, en la Figura 2.7-c, la suma de la distancia del vértice *C* con la distancia desde *C* a *D* es mayor a la distancia que ya contiene D; por lo tanto, el vértice D no se re-etiqueta. Tras inspeccionar todos los vértices adyacentes al vértice permanente y cambiar las etiquetas tentativas (de ser posible), se busca en el grafo completo el vértice etiquetado tentativamente con la menor distancia acumulada. Este vértice se hace permanente y se convierte en el nuevo nodo de trabajo para la siguiente ronda. Este procedimiento se repite hasta obtener el grafo de la Figura 2.7-f.

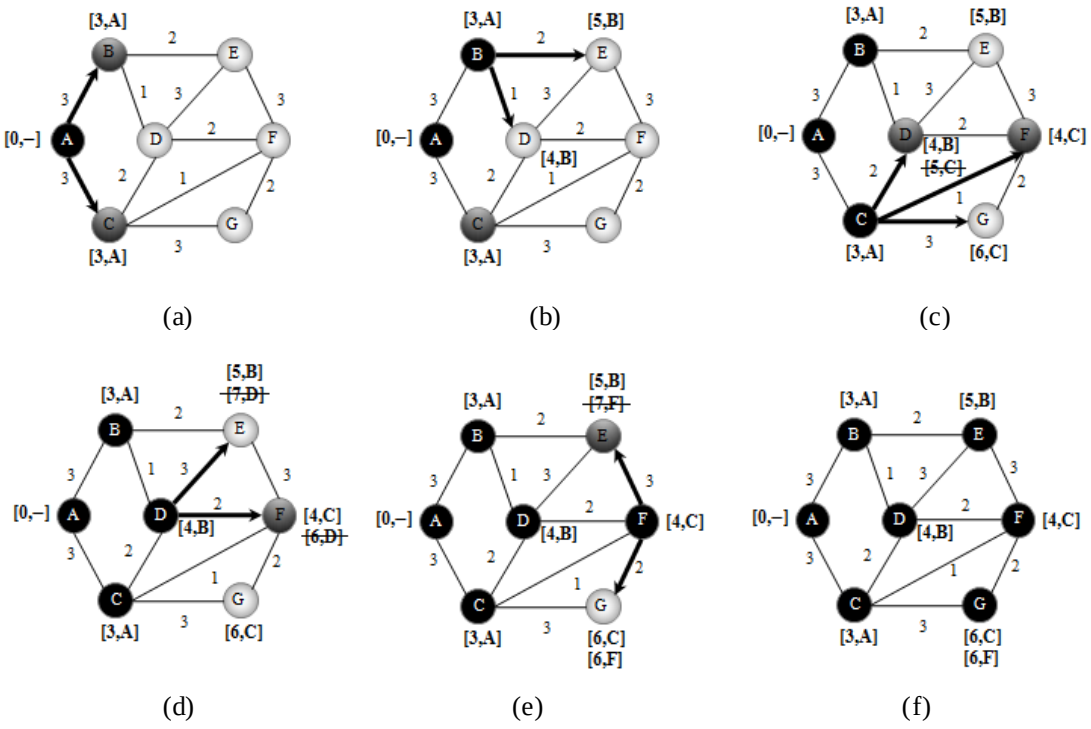


Fig. 2. 7 Ejecución del Algoritmo de Dijkstra etiquetado

La ruta se reconstruye desde el nodo final al nodo inicial; por ejemplo, si se desea conocer la ruta más corta desde *A* a *G* basta con verificar los nodos de procedencia. De acuerdo con la Figura 2.7-f el vértice *G* cuenta con dos rutas más cortas de valor de 6. En la primera opción, se observa que el vértice *G* procede del C y a su vez, el vértice C procede del *A*, por lo que la ruta final es *ACG*. La segunda opción se obtiene de manera similar, dando como resultado la ruta *ACFG*.

2.4 Fundamentos de Lógica Difusa

El término de lógica difusa fue introducido por primera vez por Lofti Zadeh alrededor de 1965 en su artículo *Fuzzy sets*. En éste, se presentaron nuevas expresiones matemáticas que contravenían a los conceptos de la lógica clásica, la cual trabaja con información definida y precisa.

La lógica difusa permite a los sistemas tratar con información que no es exacta; es decir, información que contiene un alto grado de imprecisión como por ejemplo: estatura alta, temperatura media, velocidad baja, etc. [21].

Para comprender la diferencia entre la lógica clásica y la lógica difusa, Zadeh utilizó el ejemplo de la clasificación de hombres según su estatura. De acuerdo a la Figura 2.8, la lógica clásica considera a los hombres "Bajos" si su estatura es menor a 1.80 m. y "Altos" si su estatura es mayor o igual a 1.80 m. Sin embargo, no es muy lógico decir que un hombre de estatura 1.79 es bajo y que otro de 1.80 es alto, cuando su estatura solo difiere en un centímetro. Para solucionar este problema, la lógica difusa se basa en comprender los cuantificadores de nuestro lenguaje, tales como: mucho, muy, poco, medio, etc.

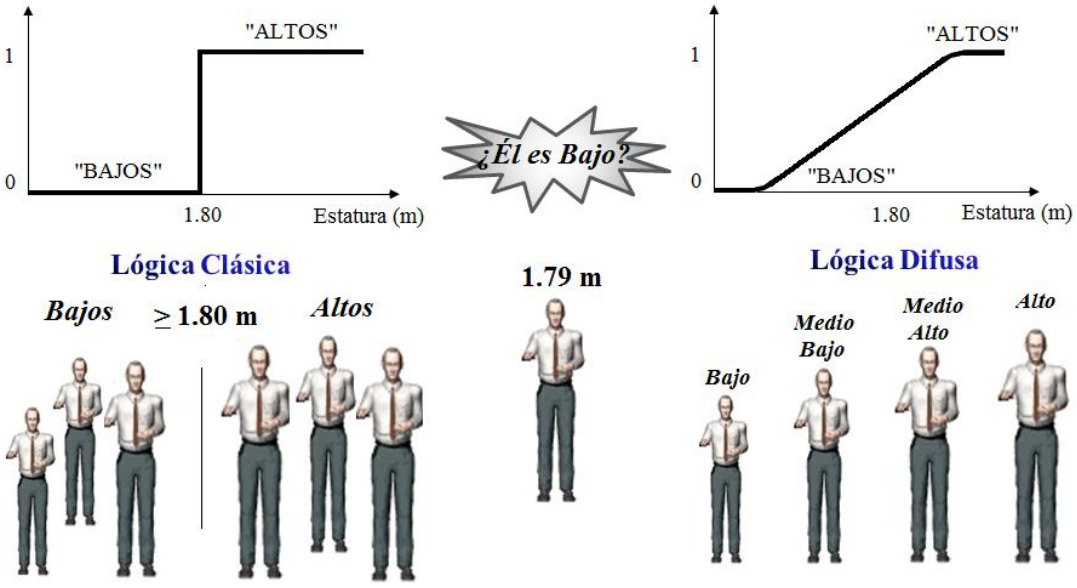
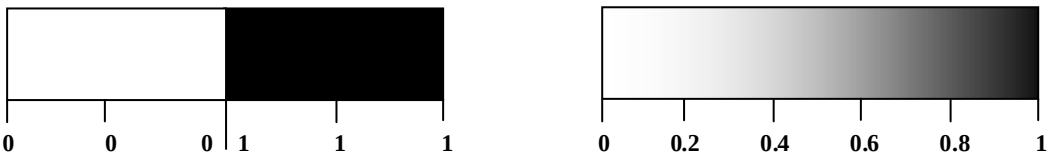


Fig. 2. 8 Diferencia entre la lógica clásica y la lógica difusa

2.4.1 Conjuntos difusos y funciones de membresía

La teoría de conjuntos difusos parte de la similitud con los conjuntos clásicos en los cuales la función de membresía se restringe a tomar un solo valor de 0 ó 1. En los conjuntos difusos, la función de membresía puede adquirir valores entre 0 a 1. La Figura 2.9 muestra la diferencia entre el rango de valores de la lógica booleana y la lógica difusa [22, 23].



(a) Conjunto clásico (b) Conjunto Difuso

Fig. 2. 9 Diferencia entre el rango de valores

Definición 2.4. Conjuntos difusos y funciones de membresía. Sea X un espacio de objetos y x los elementos genéricos de X, entonces un conjunto difuso A en X se define como una serie de pares ordenados $A = \{(x, \mu_A(x)) \mid x \in X\}$, donde $\mu_A(x)$ es llamado la función de membresía (FM) para el conjunto difuso A. La FM mapea cada elemento de X a un grado de membresía entre 0 y 1 (incluido). [23].

El concepto de conjunto difuso se encuentra asociado con un determinado valor lingüístico, el cual se define por una etiqueta, palabra o adjetivo. Una variable lingüística se caracteriza por $(X, T(x), x)$ en donde x es el nombre de la variable; $T(x)$ son los valores lingüísticos que pueden tomar la variable; X es el conjunto universal.

Por ejemplo, considere que X = "Estatura", entonces podemos definir los conjuntos difusos "Baja", "Media" y "Alta" que son caracterizados por las FM's. Por lo que la variable "Estatura" es representada por {"Estatura", (Baja, Media, Alta), R+} y se ilustra en la Figura 2.10. Si un hombre mide 1.58 m es de "Estatura Baja" con una función de membresía de grado 0.2 ó es de "Estatura Media" con una función de membresía de grado 0.8.

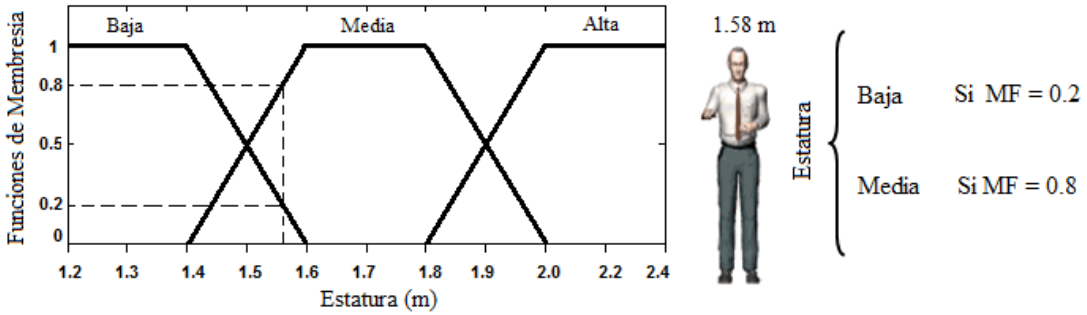


Fig. 2. 10 Ejemplo de las FM's con valores lingüísticos "Baja", "Media" y "Alta"

2.4.2 Operaciones con conjuntos difusos

Las operaciones básicas en los conjuntos clásicos son: unión, intersección y complemento. Similarmente, los conjuntos difusos también se pueden unificar, intersectar y complementar.

Definición 2.5. Unión, Intersección y Complemento Difuso. Sean A y B conjuntos difusos definidos en universo mutuo $\mathcal{U}$ [24].

La unión difusa de A y B es:

$$A \cup B \equiv \left\{\langle x, \mu_{A\cup B}(x)\rangle \mid x \in \mathcal{U} \text{ y } \mu_{A\cup B}(x) = \max\left(\mu_A(x), \mu_B(x)\right)\right\} \quad (2.4)$$

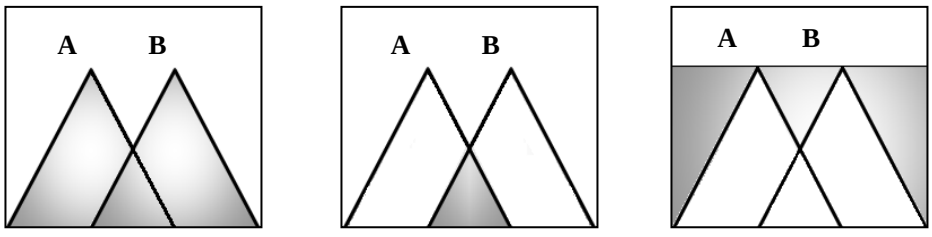
La intersección difusa de A y B es:

$$A \cap B \equiv \left\{\langle x, \mu_{A\cap B}(x)\rangle \mid x \in \mathcal{U} \text{ y } \mu_{A\cap B}(x) = \min\left(\mu_A(x), \mu_B(x)\right)\right\} \quad (2.5)$$

El complemento difuso de A es:

$$\bar{A} \equiv \{\langle x, \mu_{\bar{A}}(x)\rangle \mid x \in \mathcal{U} \text{ y } \mu_{\bar{A}}(x) = 1 - \mu_A(x)\} \quad (2.6)$$

Las propiedades y operaciones de unión, intersección y complemento se representan gráficamente en la Figura 2.11.



(a) Unión A ∪ B (b) Intersección A ∩ B (c) Complemento $\overline{A \cup B}$

Fig. 2. 11 Operaciones difusas básicas

2.4.3 Reglas difusas

Las reglas difusas IF-THEN o declaraciones condicionales difusas son expresiones de la forma IF *A* THEN *B*, donde la parte IF es el antecedente, A y B son etiquetas lingüísticas de conjuntos difusos y THEN el consecuente. Debido a su forma consistente, las reglas difusas IF-THEN son con frecuencia empleadas para capturar los modos imprecisos de razonamiento. Éstos juegan un papel escencial en la habilidad humana para fabricar decisiones en un ambiente de incertidumbre e impresición. Un ejemplo que describe un hecho simple es:

IF *x* es *A*, THEN *y* es *B*

Donde *x* y *y* son variables lingüísticas, A y B son valores lingüísticos o etiquetas que son caracterizadas por FM's [25]. Las reglas difusas pueden tener múltiples antecedentes unidos por las palabras claves AND (conjunción), OR (disyunción) ó la combinación de ambas.

2.5 Control Difuso: Principios de Operación

El objetivo principal en la ingeniería es extraer y aplicar los nuevos conocimientos acerca de cómo controlar un proceso a fin de que el sistema resultante sea confiable y seguro. El control difuso proporciona una metodología para representar, manipular e implementar el conocimiento de un humano en un sistema de control [26].

El diagrama a bloques de un sistema de control difuso se muestra en la Figura 2.12. El controlador difuso está compuesto por los siguientes cuatro elementos [26]:

1. Base de reglas ó conjunto de reglas IF-THEN.
2. Mecanismo de inferencia, también llamado motor ó máquina de inferencia.
3. Interfaz de fusificación.
4. Interfaz de defusificación.

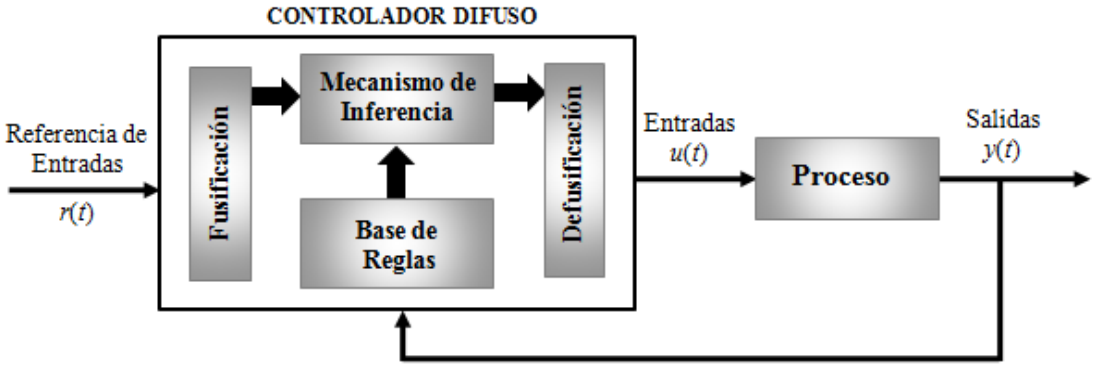


Fig. 2. 12 Arquitectura de un controlador difuso

2.5.1 Base de reglas

La base de reglas contiene las reglas lingüísticas del control. Estas reglas son del tipo IF-THEN y en algunos casos con múltiples antecedentes conectados por los operadores lógicos AND, OR y NOT. La forma general de las reglas difusas en control es:

$$\textbf{IF} \quad u_1 \text{ es } A \quad \textbf{AND} \quad u_2 \text{ es } B \quad \textbf{THEN} \quad y \text{ es } C \tag{2.7}$$

donde A, B y C son los conjuntos difusos de las variables de entrada u_1 y u_2 y de la variable de salida y.

La regla de la ecuación 2.7 tiene dos antecedentes y un consecuente. Comúnmente las variables difusas (u_1, u_2) pueden representar cantidades físicas o sistemas tales como: "temperatura", "presión", etc. y los valores lingüísticos difusos (*A*, *B*) pueden ser "caliente", "muy caliente", "frío", etc. Este tipo de reglas se conocen como reglas tipo Mamdami [27].

Otra clase de reglas difusas IF-THEN, fue propuesta por Takagi y Sugeno [28], donde los conjuntos difusos son introducidos unicamente en la parte antecedente. La ecuación 2.8 es una regla tipo Sugeno, la cual expresa la fuerza de resistencia de un objeto en movimiento [25].

$$\textbf{IF} \quad velocidad \text{ es } Alta \quad \textbf{THEN} \quad Fuerza = k * (velocidad)^2 \tag{2.8}$$

donde *Alta* en la parte antecedente es una etiqueta lingüística caracterizada por una FM. Por el contrario, la parte consecuente es descrita por una ecuación matemática de la variable de entrada: *velocidad*.

Ambos tipos de reglas difusas han sido usadas extensamente en el modelado y control de sistemas [29, 30, 31].

2.5.2 Interfaz de Fusificación

El proceso de fusificación transforma las variables de entrada $u_i \in \boldsymbol{U}_i$ en variables difusas. Para esta interfaz se deben tener definidos los rangos de variación de las variables de entrada y los conjuntos difusos asociados con sus respectivas FM's.

Definición 2.6. Fusificación. Considere $\boldsymbol{U}_i^*$ como el conjunto de todos los posibles conjuntos difusos que pueden ser definidos sobre $\boldsymbol{U}_i$. Dado $u_i \in \boldsymbol{U}_i$, la fusificación convierte u_i a conjuntos difusos denotados por $\hat{A}_i^{fuz}$ definidos en el universo del

discurso U_i. Esta transformación es producida por el operador de fusificación $\mathcal{F}$ definido por: $\mathcal{F} : U_i \rightarrow U_i^*$, donde $\mathcal{F}(u_i) = \hat{A}_i^{fuz}$ [26].

Los fusificadores son generalmente divididos en *singleton* y *no-singleton*. Un fusificador singleton tiene como soporte un solo valor u_i. La función de membresía está definida por:

$$\mu_{\hat{A}_i^{fuz}}(x) = \begin{cases} 1 & x = u_i \\ 0 & \text{para cualquier otro} \end{cases} \tag{2.9}$$

Los fusificadores no-singleton son aquellos en los cuales el soporte está en más de un punto; por ejemplo, las funciones triangulares, trapezoidales, gaussianas, etc. En éstos fusificadores, $\mu_{\hat{A}_i^{fuz}}(x) = 1$ para $x = u_i$, donde u_i puede estar en uno o más puntos, y entonces $\mu_{\hat{A}_i^{fuz}}(x)$ decrece desde 1 tanto como x se aleje de u_i [23, 32]. La Figura 2.13 muestra un ejemplo de la fusificación singleton y no-singleton.

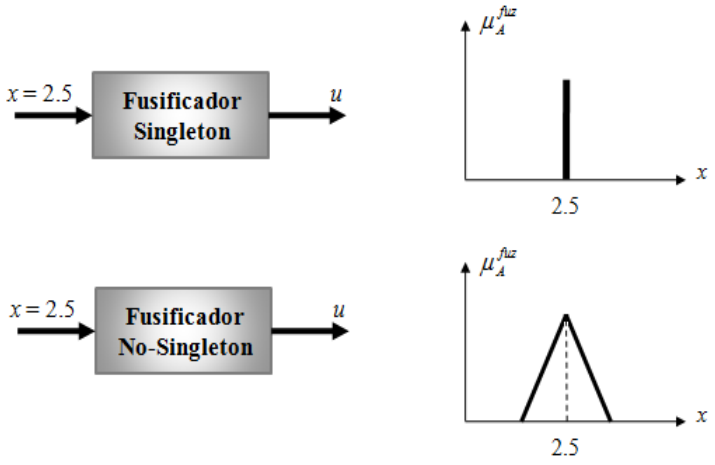


Fig. 2. 13 Ejemplos de los fusificadores singleton y no-singleton

2.5.3 Mecanismo de Inferencia

Una vez que las entradas son fusificadas, los correspondientes conjuntos difusos de entrada son pasados al mecanismo de inferencia. Este último realiza la tarea de calcular las variables de salida a partir de las variables de entrada, mediante las reglas del controlador y la inferencia difusa, entregando conjuntos difusos de salida.

El método de Mamdani es el más utilizado como mecanismo de inferencia, debido a que tiene una estructura muy simple de operaciones "min-máx.". Este método se basa en los siguientes pasos [26]:

1. Evaluación del antecedente de cada regla. Si el antecedente de una regla tiene más de un término, se aplica algún operador t-norma obteniendo un único valor de membresía.

2. Obtener la conclusión en cada regla. A partir del consecuente de cada regla y del valor del antecedente obtenido en el paso 1, se aplica un operador de implicación difuso obteniendo así un nuevo conjunto difuso. Los operadores más utilizados son el *mínimo* (min), que trunca la FM; y el producto, el cual la escala.

3. Combinación en un único conjunto difuso. Las salidas obtenidas para cada regla en el paso 2 se combinan en un único conjunto difuso (μ_z) utilizando un operador de agregación difuso. Algunos de los operadores de agregación más utilizados son el *máximo* (máx.), la *suma* o el *OR* probabilístico.

La Figura 2.14 ilustra un ejemplo del método de inferencia Mamdani, en éste se utiliza el operador *mínimo* para obtener la conclusión de cada regla y el operador *máximo* en la combinación en un único conjunto difuso.

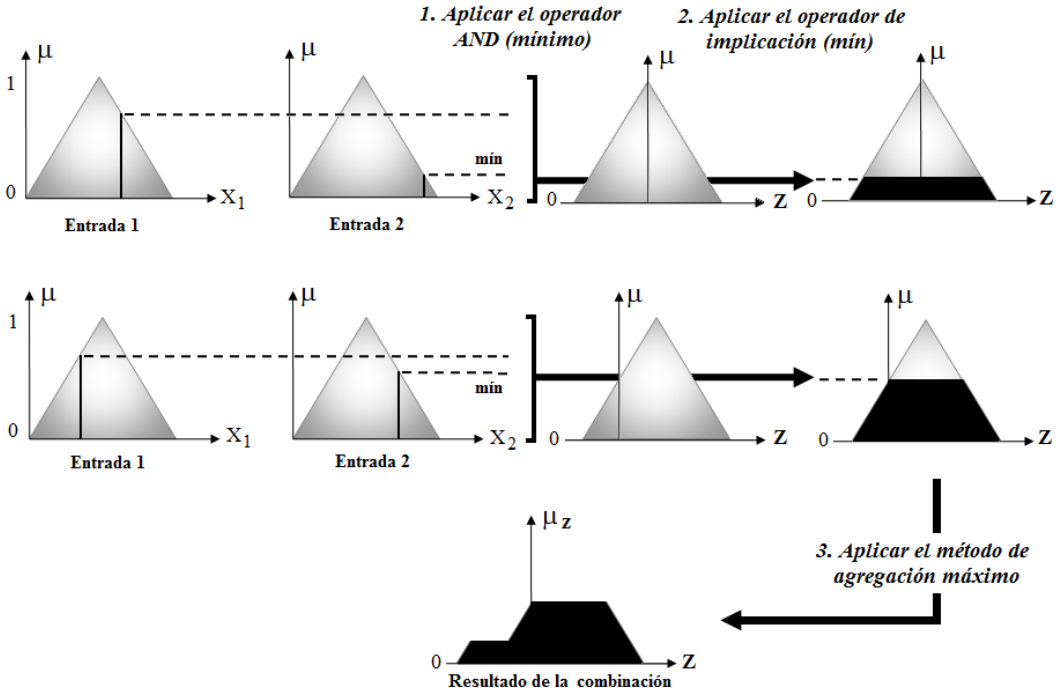


Fig. 2. 14 Método de Inferencia Mamdani (mínimo-máximo)

2.5.4 Interfaz de defusificación

Como última parte de los procesos del control difuso, se lleva a cabo la defusificación. Esta interfaz provee salidas discretas a partir de los conjuntos difusos obtenidos como resultado de la inferencia. Existen diferentes métodos de defusificación, los principales se describen a continuación [33]:

1. Método del máximo. La salida corresponde al valor para el cual la FM del conjunto de salida μ_z alcanza su máximo.
2. Media del máximo. La salida es el promedio entre los elementos del conjunto de salida, resultado de la inferencia, que tengan un grado de membresía máximo.
3. Bisectriz. La bisectriz es la línea vertical que divide el conjunto de salida en dos regiones de igual área.

4. Centro de área o centroide. Genera como salida el valor correspondiente al centro de gravedad de la FM del conjunto de salida (μ_z).

El método de defusificación más utilizado es el del centroide, dado el polígono de salida, resultado del proceso de inferencia, se calcula el centro de gravedad (z*) mediante la siguiente fórmula:

$$z^* = \frac{\int \mu_z(z)\, z\, dz}{\int \mu_z(z)\, dz} \tag{2.9}$$

Para el caso se los sistemas discretos, se calcula el centro de gravedad discretizado con la fórmula 2.10. La Figura 2.15 ilustra un ejemplo de la defusificación con centro de gravedad discreto.

$$z^* = \frac{\sum \mu_z(z_i) z_i}{\sum \mu_z(z_i)} \tag{2.10}$$

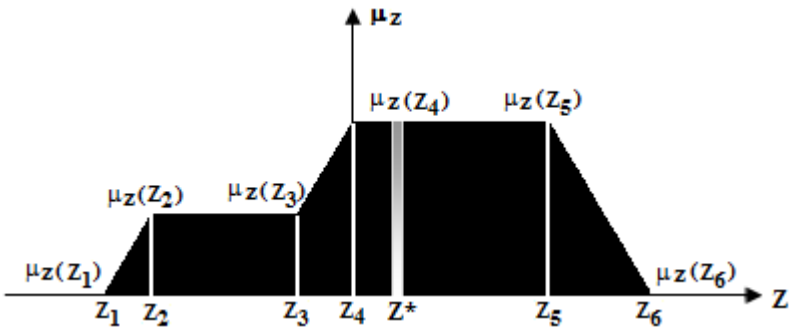


Fig. 2. 15 Método del centro de área o centroide discretizado

2.6 Lógica Difusa en la Navegación Autónoma

La meta de la robótica autónoma móvil es construir un sistema físico que pueda moverse con éxito sin la intervención humana en ambientes desconocidos; es decir, en ambientes del mundo real que no han sido diseñados específicamente para el robot. El desarrollo de técnicas para la navegación autónoma constituye uno de los mayores retos de las investigaciones actuales en robótica móvil [34].

En trabajos tradicionales de robótica móvil se han diseñado los mecanismos robóticos y el ambiente sobre el cual éste se desenvolverá. Sin embargo, diseñar el ambiente o el robot incrementa los costos, reduce la autonomía y no puede ser aplicado en todos los dominios. El principal desafío de la robótica autónoma actual es construir programas que ejecuten confiablemente tareas complejas a pesar de la incertidumbre del ambiente [35].

Desde la década de los ochenta, la lógica difusa ha jugado un papel muy importante en el control de robots móviles. En 1985, Sugeno y Nashida reportaron el primer trabajo que utilizó el control difuso en la robótica móvil. En éste, se desarrolló un controlador difuso capaz de conducir un robot móvil a lo largo de un camino delimitado por dos paredes [36]. Poco tiempo después, se desarrolló un controlador difuso para la evasión de obstáculos, el cual se basaba en un algoritmo para obtener información acerca de las áreas libres y ocupadas encontradas frente al robot móvil [37]. En la actualidad, estas dos fundamentales tareas se siguen implementando en la mayoría de las aplicaciones de la robótica móvil.

En general, la lógica difusa se emplea en el diseño de unidades de comportamiento individual, tales como evitar obstáculos, seguir trayectorias o a alguna característica del ambiente (paredes, el acotamiento del camino, líneas negras ó blancas en el piso, etc.). Los comportamientos difusos pueden ser sintetizados en un conjunto de reglas tipo IF-THEN, de tal forma que el conocimiento experto este reflejado en términos lingüísticos de fácil comprensión.

Los controladores difusos son típicamente diseñados para considerar una sola meta. Si se desea considerar dos o más metas, se tienen dos opciones. En la primera opción, se puede escribir una serie de reglas complejas cuyos antecedentes consideran ambas metas simultáneamente. Por ejemplo, si las dos metas son encaminar hacia una ruta y evitar los obstáculos, las reglas difusas son de la forma general:

$$\begin{cases} \textbf{IF } \textit{condición de enrutamiento}_1 \textbf{ AND } \textit{condición de evasión}_1 \textbf{ THEN } \textit{acción}_{11} \\ \textbf{IF } \textit{condición de enrutamiento}_1 \textbf{ AND } \textit{condición de evasión}_2 \textbf{ THEN } \textit{acción}_{12} \\ \cdot \\ \cdot \\ \cdot \\ \textbf{IF } \textit{condición de enrutamiento}_n \textbf{ AND } \textit{condición de evasión}_m \textbf{ THEN } \textit{acción}_{nm} \end{cases}$$

La segunda opción consiste en escribir dos conjuntos de reglas simples, cada uno de ellos especificando cada meta y combinando sus salidas de alguna manera. La forma general de estas reglas es:

$$\begin{cases} \textbf{IF } \textit{condición de enrutamiento}_1 \textbf{ THEN } \textit{acción de enrutamiento}_1 \\ \cdot \\ \cdot \\ \cdot \\ \textbf{IF } \textit{condición de enrutamiento}_n \textbf{ THEN } \textit{acción de enrutamiento}_n \end{cases}$$

$$\begin{cases} \textbf{IF } \textit{condición de evasión}_1 \textbf{ THEN } \textit{acción de evasión}_1 \\ \cdot \\ \cdot \\ \cdot \\ \textbf{IF } \textit{condición de evasión}_m \textbf{ THEN } \textit{acción de evasión}_m \end{cases}$$

Este tipo de reglas se pueden ver como la especificación de planes de acción para el robot, en la forma de *situación → acción*. Por ejemplo, el conjunto de reglas listadas en la Tabla 2.5 constituyen el plan para alcanzar el cuarto-4 ilustrado en la Figura 2.16.

Tabla 2. 5 Conjunto de reglas difusas para ejecutar un plan completo

IF Obstáculo	**THEN** Evitar
IF (¬ Obstáculo ^ dentro (Corr-1) ^ ¬ dentro (Corr-2))	**THEN** Avanzar (Corr-1)
IF (¬ Obstáculo ^ dentro (Corr-2) ^ ¬ cerca (Puerta-4))	**THEN** Avanzar (Corr-2)
IF (¬ Obstáculo ^ cerca (Puerta-4) ^ ¬ dentro (Cuarto-4))	**THEN** Atravesar (Puerta-4)
IF dentro (Cuarto-4)	**THEN** Detener

El plan se ejecuta de la siguiente manera: si el robot se encuentra en el corredor 1 (Corr-1) avanza hasta alcanzar el corredor 2 (Corr-2); entonces continua en el corredor 2 hasta encontrarse con la puerta-4 y cruzarla [38].

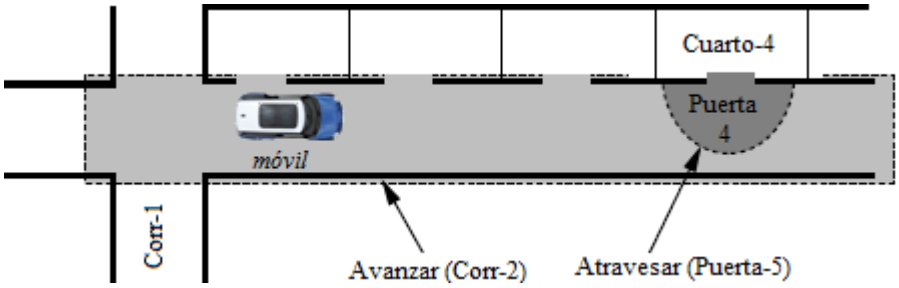


Fig. 2. 16 Ambiente sobre el cual navega un robot móvil

En general, la lógica difusa posee características que son particularmente atractivas para solucionar los problemas expuestos por la navegación de robots autónomos. La combinación de la lógica difusa con máquinas de aprendizaje logra que los sistemas ejecuten más que las simples tareas descritas previamente [39, 40].

En este capítulo se han descrito los conceptos básicos que son involucrados en la navegación autónoma de robots móviles. Se describieron los sistemas SLAM y los diferentes algoritmos de enrutamiento por la ruta más corta. Se expuso la razón por la cual el algoritmo de Dijkstra es el mejor candidato a implementar en este trabajo. Por último se definieron los conceptos básicos de la lógica difusa y las partes de un controlador difuso; así como también la importancia que tienen en las aplicaciones de la navegación autónoma.

Capítulo 3

Diseño e Implementación del sistema

En este capítulo se describe el procedimiento del diseño e implementación del sistema de navegación. El sistema se desarrolla en Matlab utilizando una interfaz gráfica para simular la posición y movimiento del móvil.

3.1 Descripción del sistema

El sistema de navegación consiste de la arquitectura Lego NXT conectada inalámbricamente vía Bluetooth a una computadora portátil. Dicha arquitectura móvil se encuentra equipada con una cámara web y un sensor ultrasónico.

Como requerimiento inicial, el sistema necesita la información referente a la posición de los obstáculos. Para ello, se le ofrece al usuario dos opciones: en la primera, el usuario especifica manualmente el número y posición de obstáculos. En la segunda opción, el sistema opera en forma automática, a través de la captura de la información visual del entorno real en el que se desplazará el robot dentro de un entorno controlado.

Fig. 3. 1 Representación esquemática del sistema

La imagen adquirida mediante una cámara web es importada hacia el ambiente de trabajo de Matlab por medio de la funciones de adquisición de imágenes. Dicha cámara se encuentra montada de tal manera que logre captar todo el ambiente real, tal como se muestra en el esquema de la Figura 3.1.

Una vez que el sistema obtiene información visual relacionada con el entorno, localización del robot, obstáculos, inicio y fin del recorrido, se genera y entrega en forma inalámbrica al robot móvil el algoritmo de enrutamiento para que sea capaz de recorrer la ruta más corta libre de colisiones.

La etapa de corrección de la ruta se incorpora en la forma de un control difuso de lazo cerrado con retroalimentación en base a la información visual proporcionada por la cámara montada sobre el robot. El diagrama de flujo del sistema se muestra en la Figura 3.2.

Por último, con la finalidad de construir un algoritmo en el cual el robot sea capaz de explorar el entorno, buscar las colisiones presentes y reportar la localización de los mismos. El diagrama de flujo del sistema operando de esta forma se describe en la Figura 3.3.

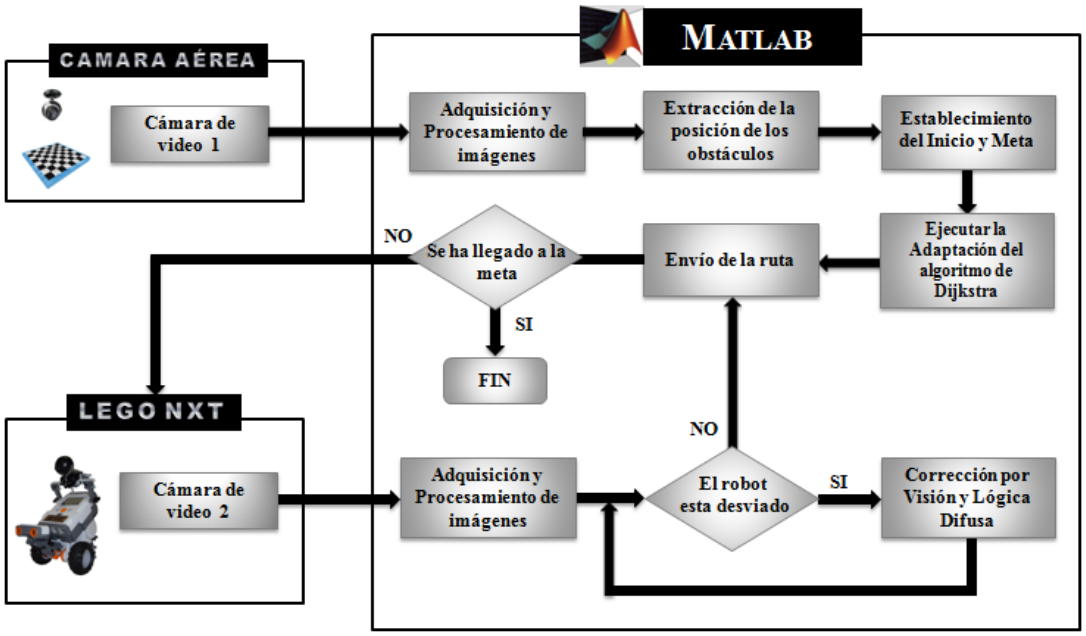


Fig. 3. 2 Diagrama de flujo del sistema operando en forma automática 1, a través de la captura de la información visual del entorno real

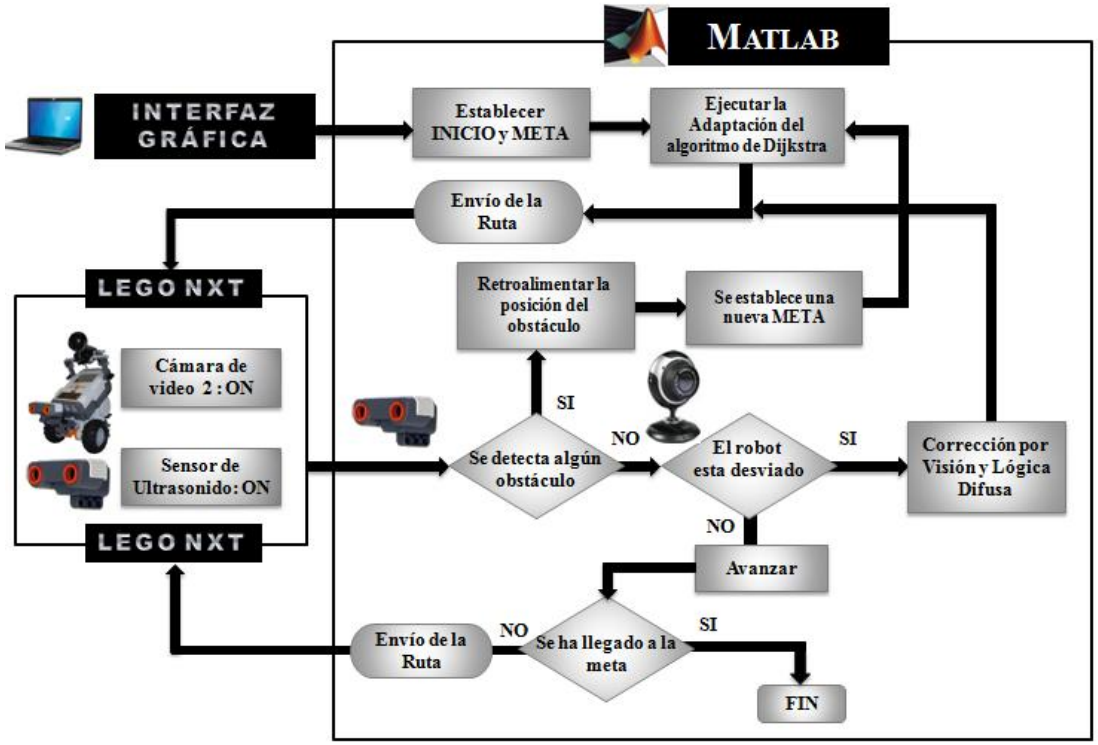


Fig. 3. 3 Diagrama de flujo del sistema operando en forma automática 2, a través de la exploración del entorno

Las dimensiones reales del sistema se describen a través de la Figura 3.4 y de la Tabla 3.1.

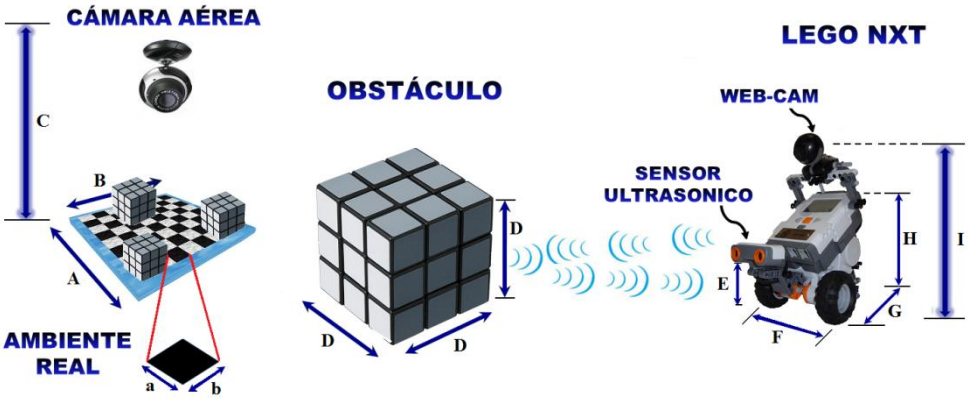


Fig. 3. 4 Descripción de las dimensiones reales del sistema

Tabla 3. 1 Descripción de las dimensiones reales del sistema

Variable	Descripción	Dimensión	Variable	Descripción	Dimensión
A	Lado A del Entorno	1 m	**E**	Altura al Sensor Ultrasónico	7 cm
B	Lado B del Entorno	1 m	**F**	Acho del Lego NXT	15 cm
a, b	Lados *a* y *b* del cuadro	20 cm	**G**	Largo del Lego NXT	18 cm
C	Altura de la Cámara Aérea	2.25 m	**H**	Altura del Lego NXT	16 cm
D	Lado del cubo (Obstáculo)	15 cm	**I**	Altura al lente de la Cámara	19 cm

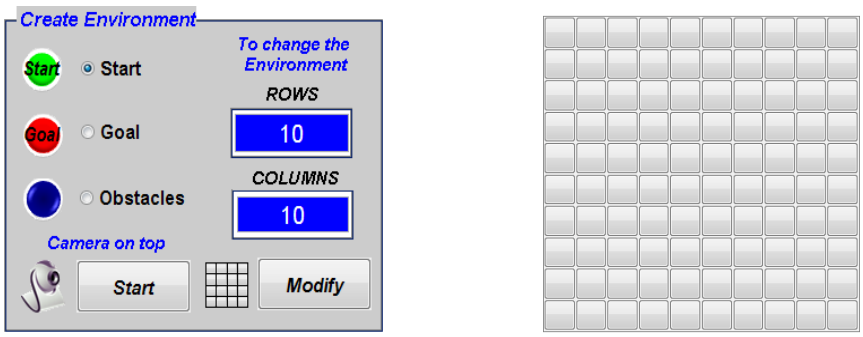
3.2 Representación del entorno de navegación

Con la finalidad de verificar que los movimientos del robot móvil sean correctos, se diseña un ambiente virtual para el sistema de navegación. Esta representación gráfica está basada en el entorno real controlado y es desarrollada utilizando GUIDE (*Graphical User Interface Development Environment*) de Matlab.

El ambiente virtual consiste en una matriz de cuadros interconectados entre sí. Esta matriz cambia de dimensiones según la decisión del usuario. Tal como se mencionó en la sección 3.1; antes de cualquier ejecución, el sistema requiere la

información referente a la posición inicial del móvil, de los obstáculos y del objetivo.

Con respecto a la posición de los obstáculos, el sistema ofrece al usuario dos opciones. En la primera opción, que puede ser utilizada como una etapa de prueba, el usuario especifica de forma manual el número y posición de los obstáculos. En la segunda opción, el sistema opera en forma automática, a través de la captura de la información visual del entorno real en el que se desplazará el robot dentro de un entorno controlado. Estas opciones se ilustran en la Figura 3.5.



(a) Opciones para crear el entorno virtual (b) Ambiente virtual de 10x10

Fig. 3. 5 Creación del ambiente virtual en Matlab

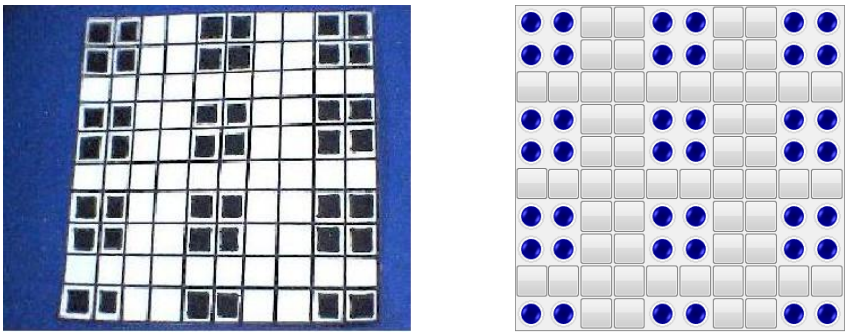
En ambas opciones, el usuario tiene la libertad de elegir la posición del nodo de INICIO y del nodo META. Cabe mencionar, que únicamente hay un nodo INICIO y un nodo META en todo el ambiente, por lo que no hay múltiples caminos.

3.2.1 Captura del ambiente real para la detección automática de obstáculos

Cuando el sistema opera de manera automática, la información visual del entorno real controlado es requerida por el sistema. Para lograrlo, se instala una cámara de tal manera que logre captar todo el ambiente real.

La imagen adquirida mediante la cámara web es importada hacia el ambiente de trabajo de Matlab por medio de la funciones de adquisición de imágenes. Este proceso se ejecuta con la finalidad de extraer la posición de los obstáculos y plasmarlos en el ambiente virtual.

Con fines experimentales, se realizó un modelo a escala (1:20 cm) del entorno real, de tal manera que una cámara web con una resolución de 352x288 a una distancia de 25 cm logre captar toda la imagen, tal como se muestra en la Figura 3.6.



(a) Ambiente captado por la cámara web (b) Resultado de la ejecución

Fig. 3. 6 Captura, extracción y posicionamiento de los obstáculos en el modelo a escala

Con el objetivo de eliminar el área que no corresponde al entorno, además de extraer la posición y número de los obstáculos presentes en el mismo, se lleva a cabo una serie de operaciones de procesamiento sobre la señal de video capturada. La tabla 3.2 presenta en pseudocódigo el algoritmo utilizado en Matlab.

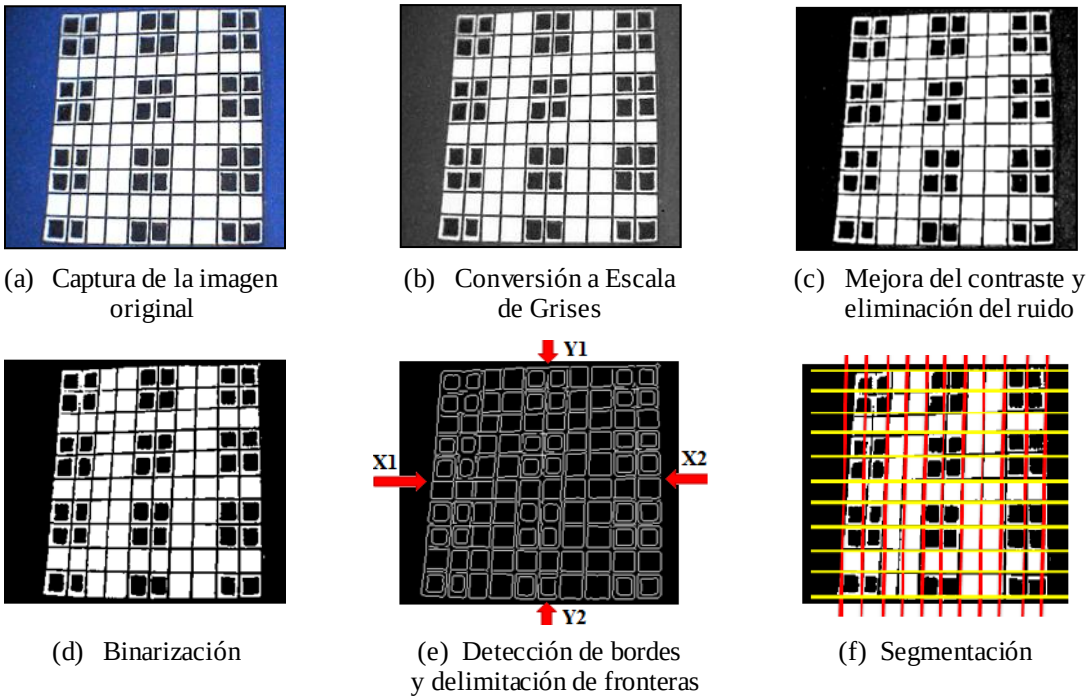
La ejecución de este algoritmo se ilustra en la Figura 3.7. Las líneas 1-5 detallan el procesamiento que se hace sobre la imagen capturada por la cámara web. La línea 6 describe la eliminación del área que no corresponde al entorno.

Tabla 3. 2 Pseudocódigo del algoritmo de Detección y Posicionamiento de los obstáculos

Algoritmo 3.2 Procesamiento de Imagen y Detección de Obstáculos

DETECCIÓN Y POSICIONAMIENTO DE OBSTÁCULOS

```
1:  Captura de un cuadro de la imagen de video.
2:  Conversión a escala de grises.
3:  Mejora del contraste de la imagen y eliminación del ruido.
4:  Binarización de la imagen.
5:  Detección de bordes.
6:  Delimitación de fronteras del ambiente dentro de la imagen de bordes.
7:  Segmentación del ambiente en una rejilla sobre la imagen binarizada.
8:  Detección y posicionamiento de obstáculos
9:  for cada segmento i contenido en la imagen.
10:     If moda (segmento(i)) = 0 then
11:         Nodo (i) = Obstáculo
12:     else
13:         Nodo (i) = Nodo libre
14:     end if
15: end for
```



(a) Captura de la imagen original

(b) Conversión a Escala de Grises

(c) Mejora del contraste y eliminación del ruido

(d) Binarización

(e) Detección de bordes y delimitación de fronteras

(f) Segmentación

Fig. 3. 7 Procesamiento de la imagen del entorno real

La eliminación del área sobrante de la Figura 3.7-a consiste en tomar la imagen de bordes mostrada en la Figura 3.7-e y detectar el punto medio de los cuatro lados de la imagen. Una vez establecidos estos puntos, se realiza un barrido horizontal y vertical sobre los cuatro lados de la imagen. Esto último es con la finalidad de detectar la ubicación del pixel donde se presenta la transición del color negro (0) a blanco (1). De esta manera, se cuentan con cuatro valores diferentes: X_1, X_2, Y_1 y Y_2. El tamaño real del modelo de la interfaz se determina por:

$$Ancho = X_2 - X_1 \tag{3.1}$$

$$Altura = Y_2 - Y_1 \tag{3.2}$$

Los valores de *Ancho* y *Altura* obtenidos por las ecuaciones 3.1 y 3.2 son esenciales en la segmentación de la imagen. El proceso de segmentación descrito en la línea 7 de la Tabla 3.2 consiste en tomar los valores *Ancho* y *Altura* y dividirlos entre el número de columnas y filas, respectivamente.

$$a = \frac{Ancho}{No.\ de\ columnas\ del\ ambiente} \tag{3.3}$$

$$b = \frac{Altura}{No.\ de\ filas\ del\ ambiente} \tag{3.4}$$

Estos nuevos valores *a* y *b* son las dimensiones de cada uno de los segmentos imaginarios que se crean sobre la imagen binarizada. La Figura 3.7-f ilustra un ejemplo del proceso de segmentación.

Por último, sobre cada uno de estos segmentos se obtiene la medida modal para indicar que valor se repite más veces. Si el valor 0 es el que más predomina, se establece que el segmente es un obstáculo. De lo contrario, se asigna como una casilla libre.

Cabe mencionar que de acuerdo a los resultados de las pruebas que se describen en el capítulo 4, este algoritmo es muy sensible a las condiciones de iluminación. El exceso de brillo en la imagen capturada provoca que uno ó varios obstáculos sean eliminados en el proceso de binarización, por lo que se reportan posiciones libres en nodos donde en realidad se encuentran obstáculos. Por otra parte, las sombras que se generan por los cambios de iluminación provocan que el algoritmo establezca obstáculos sobre las posiciones libres.

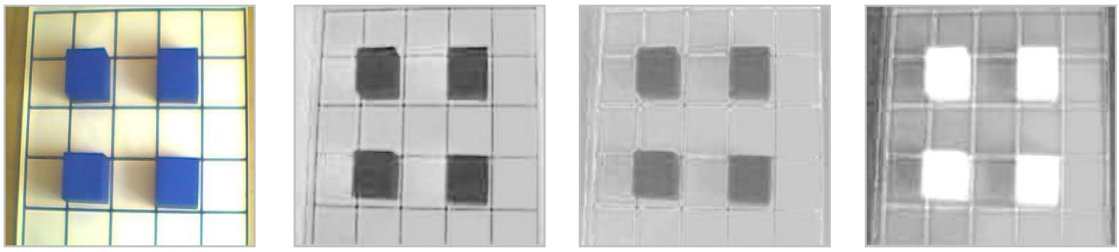
Debido a lo anterior, es necesario modificar el algoritmo 3.2 para volverlo tolerante a las sombras y a los cambios en la iluminación. Los modelos de color HSI (Hue, Saturation, Insensity) y el c_1, c_2, c_3 son destacables por ser insensibles a los efectos del cambio en la iluminación [41]. Sin embargo, se ha comprobado que el modelo HSI no es efectivo en la eliminación de las sombras; mientras que el c_1, c_2, c_3 ha demostrado ser el mejor modelo de color invariante a las sombras e iluminación [42]. Este modelo está definido por:

$$c_1(R,G,B) = arctan\left(\frac{R}{max\,(G,B)}\right) \tag{3.5}$$

$$c_2(R,G,B) = arctan\left(\frac{G}{max\,(R,B)}\right) \tag{3.6}$$

$$c_3(R,G,B) = arctan\left(\frac{B}{max\,(R,G)}\right) \tag{3.7}$$

Como puede apreciarse en la Figura 3.8, el plano c_1 es el más efectivo en la eliminación de la sombra y además resalta a los obstáculos. Por tal motivo, la conversión a escala de grises de la imagen original, descrita en la Tabla 3.2, se reemplaza por la obtención del plano c_1.



(a) Imagen original (b) Plano c_1 (c) Plano c_2 (d) Plano c_3

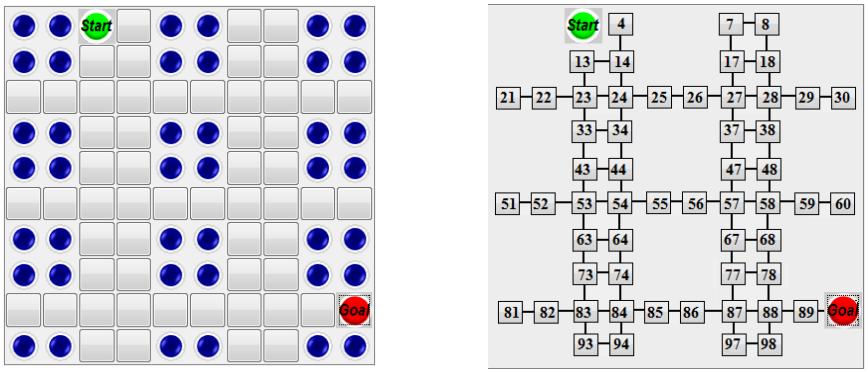
Fig. 3. 8 Planos de color c1, c2 y c3

3.3 Implementación del Algoritmo de Enrutamiento

Una vez que el sistema obtiene la información visual relacionada con el entorno, localización del robot, obstáculos, inicio y fin del recorrido, se requiere que la computadora genere y entregue en forma inalámbrica al robot móvil un algoritmo de enrutamiento para que sea capaz de recorrer la ruta más corta libre de colisiones.

Existen diversos algoritmos de enrutamiento por la ruta más corta; sin embargo, se eligió el Algoritmo de Dijkstra por diversas razones. En primer lugar, las numerosas e importantes aplicaciones existentes del mismo hacen de él uno de los más populares en la ciencia computacional, aumentando así la importancia de su aprendizaje; y por otra parte, la facilidad de su aplicación, ya que requiere un bajo nivel de procesamiento.

El algoritmo de Dijkstra se ejecuta sobre la matriz de cuadros presente en el ambiente virtual. Para adaptar el Algoritmo de Dijkstra es necesario asociar la matriz de cuadros con un grafo. Tomando en cuenta los requerimientos iníciales, el grafo se genera extrayendo únicamente los nodos válidos de la matriz de cuadros y eliminando los obstáculos. En la Figura 3.9 se puede apreciar el ambiente gráfico simulado y la estructura del grafo.



(a) Entorno gráfico simulado (b) Estructura del grafo

Fig. 3. 9 Asociación de la matriz de cuadros con un grafo

Como puede observarse en la Figura 3.9-b, a cada nodo válido se le asigna la posición numérica que ocupa dentro de la matriz de cuadros. Los obstáculos son anulados con el valor numérico 0 designado a la posición de cada de ellos. Los nodos *Start* (Inicio) y *Goal* (Meta) son identificados con las coordenadas correspondientes a su ubicación. La distancia entre cada cuadro es considerada de 1 unidad.

El algoritmo que genera el grafo se describe en la Tabla 3.3. Este consiste en crear una matriz *Grafo* de *n* filas por ocho columnas; donde *n* corresponde al número total de nodos contenidos en el ambiente gráfico simulado y cada columna corresponde a la posición de los nodos circundantes al nodo analizando.

Tabla 3. 3 Pseudocódigo del algoritmo para generar el grafo

Algoritmo 3.3 Grafo

GENERACIÓN DEL GRAFO (*n, Grafo*)

1: **for** cada nodo *n* contenido en el entorno gráfico *E*: $n \in E$
2: **if** $n = 0$ **then**
3: *Grafo* (*n*) = [0, 0, 0, 0, 0, 0, 0, 0]
4: **else**
5: *Grafo* (*n*) = [*n*Arriba, *n*ArribaDerecha, *n*Derecha, *n*AbajoDerecha, *n*Abajo, *n*AbajoIzquierda, *n*Izquierda, *n*ArribaIzquierda]
6: **end if**
7: **end for**

De acuerdo con la Tabla 3.3, si el nodo analizado es el 24 de la Figura 3.9-b, en la fila número 24 de matriz *Grafo* se almacenan los siguientes valores: [14, 0, 25, 0, 34, 33, 23, 13]. Si el nodo analizado *n* es un obstáculo, a cada columna se le asigna el valor 0.

Una vez que se construye el grafo se prosigue a ejecutar el algoritmo de Dijkstra con las modificaciones necesarias para adaptarlo a este entorno. La Tabla 3.4 presenta en pseudocódigo el algoritmo implementado en el sistema. El algoritmo requiere como entrada el nodo Inicio y el nodo Meta, la matriz *M* que contenga a cada uno de los nodos presentes en el ambiente gráfico y la matriz *Grafo*.

El algoritmo devuelve una matriz *Dijkstra* de *n* filas por tres columnas, donde *n* es el número total de nodos contenidos en el ambiente gráfico y las columnas corresponden a las variables: *distancia*, *NP* (nodo procedente) y *flag* (bandera). Esta última variable se incluye con la finalidad de verificar cuáles nodos han sido analizados.

Las líneas 1-6 describen el proceso de identificación de los obstáculos. En la matriz Dijkstra, por cada obstáculo encontrado se activa la variable *flag* en la posición numérica correspondiente al nodo que contenga dicho obstáculo y las variables *distancia* y *NP* son infinitas, de ésta forma los obstáculos son anulados. El proceso descrito en las líneas 7-9 consiste en identificar al nodo Inicio (Start) como el primer nodo permanente, por lo que su bandera *flag* es activada.

Antes de ejecutar las adaptaciones al algoritmo de Dijkstra es necesario direccionar el escaneo de la matriz de nodos *M* descrito en las líneas 10-13. Para llevar a cabo éste proceso es necesario identificar la ubicación de los nodos Inicio (Start) y Meta (Goal). Éstos últimos se localizan a través de coordenadas, las cuales se almacenan en las variables X_1, Y_1, X_2 y Y_2. Se realiza la comparación de dichas variables y dependiendo del resultado se establecen cuatro tipos de barridos: *derecha-*

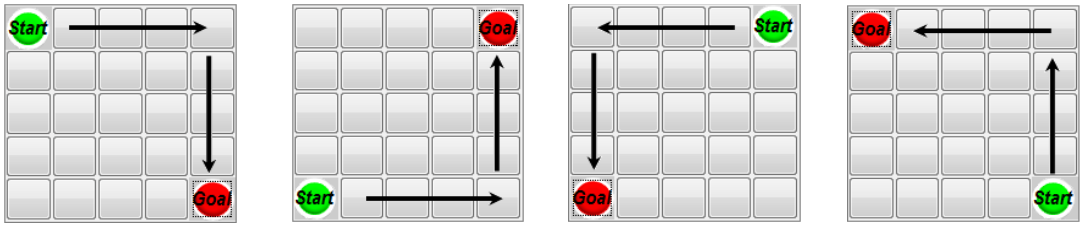
abajo, derecha-arriba, izquierda-abajo e *izquierda-arriba*, tal como se muestra en la Figura 3.10.

Tabla 3. 4 Pseudocódigo del algoritmo para generar el grafo

EJECUCIÓN DEL ALGORITMO DE DIJKSTRA

Entrada: requiere un nodo origen y meta, la matriz de nodos *M* y la matriz *Grafo*
Salida: devuelve una matriz *Dijkstra* de *n* filas y 3 columnas, donde *n* es el número total de nodos contenidos en el ambiente gráfico. Las columnas corresponden a las variables: distancia *(distancia)*, nodo procedente *(Np)* y bandera *(flag)*.

1: Localización de los obstáculos y plasmarlos en la matriz Dijkstra
2: **for** cada nodo *n* contenido en el entorno gráfico *E*: $n \in E$
3: **if** *Grafo* (*n*) = [0, 0, 0, 0, 0, 0, 0, 0] **then**
4: *Dijkstra (n).flag* = 1, *Dijkstra (n).distancia* = ∞, *Dijkstra (n).NP* = ∞
5: **end if**
6: **end for**
7: Establecer el primer nodo permanente
8: *Np = Nodo Inicio*
9: *Dijkstra (Np).flag* = 1
10: Direccionar el barrido de la matriz de nodos *M*:
11: (X_1, Y_1) = *Nodo Inicio*
12: (X_2, Y_2) = *Nodo Meta*
13: Se comparan las cuatro variables: X_1, Y_1, X_2, Y_2 y se direcciona el barrido:
14: Ejecución de las adaptaciones al Algoritmo de Dijkstra
15: **for** todos los elementos *n* de la matriz *Dijkstra*
16: **if** *Dijkstra (n).flag* ≠ 1 **then**
17: Arriba, Abajo, Izquierda, Derecha, ArribaDerecha, AbajoDerecha,… ArribaIzquierda, AbajoIzquierda = Grafo (*n*). Arriba, Abajo,… AbajoIzquierda
18: **end if**
19: **if** cada una de las variables (Arriba, Abajo,… AbajoIzquierda) = 0 **then**
20: por cada variable se asigna *distancia* $_{\text{Arriba, Abajo,… AbajoIzquierda}}$ = ∞
21: **else**
22: **if** *Dijkstra (*Arriba, Abajo,… AbajoIzquierda*).flag* = 1 **then**
23: *distancia* $_{\text{Arriba, Abajo,… AbajoIzquierda}}$ = *Dijkstra (*Arriba, Abajo,… AbajoIzquierda*).distancia*
24: **else**
25: *distancia* $_{\text{Arriba, Abajo,… AbajoIzquierda}}$ = ∞
26: **end if**
27: **end if**
28: **if** por lo menos 2 distancias (*distancia* $_{\text{Arriba, Abajo,… AbajoIzquierda}}$) son diferentes **then**
29: Se comparan las 8 distancias y se halla la menor. Se selecciona el nodo que se contenga esta distancia encontrada.
30: **do RELAJACIÓN MODIFICADA**
31: *Dijkstra(n).distancia* = *distancia* $_{\text{Arriba, Abajo,… AbajoIzquierda}}$ + 1
32: *Dijkstra(n).Np* = *n* $_{\text{Arriba, Abajo,… AbajoIzquierda}}$
33: *Dijkstra(n).flag* = 1
34: *Np = n*
35: **end if**
36: **end for**



(a) Derecha-Abajo (b) Derecha-Arriba (c) Izquierda-Abajo (d) Izquierda-Arriba

Fig. 3. 10 Tipos de direccionamiento de la matriz M

La rutina descrita en las líneas 14-37 tiene como finalidad unir aquellos nodos válidos; es decir, aquellos que no sean obstáculos y cuya bandera *flag* haya sido activada. Cabe mencionar, que cuando la ejecución inicia, el único nodo válido con el *flag* activado es el nodo Inicio y las variables *distancia* y *NP* son cero.

De acuerdo con lo anterior, éste proceso da inicio cuando el escaneo de la matriz *M* se posiciona en un algún nodo circundante al nodo Inicio. Suponga que el escaneo está situado en el nodo 4 de la Figura 3.9-b; haciendo uso de la matriz *Grafo*, la rutina verifica cuáles son los nodos circundantes al nodo 4 y con la ayuda de la matriz *Dijkstra* se inspecciona si alguno de éstos nodos tiene activa la bandera *flag*. Dado que el nodo Inicio es el único que cumple dichas condiciones se procede a guardar en la matriz *Dijkstra*, fila 4, una distancia igual a 1, el valor numérico del nodo Inicio como nodo procedente y un 1 en *flag* para indicar que el nodo ha sido analizado.

En lo que corresponde a los demás nodos circundantes al nodo 4 que no sean obstáculos y cuya bandera *flag* no está activada se les asigna una distancia igual a infinito. La línea 29 describe la posibilidad de que algún nodo pueda elegir unirse a más de un nodo; es decir, que más de un nodo circundante a él tenga activado el *flag*. En este caso, el proceso entra en una nueva rutina para determinar cuál de estos posibles nodos tiene la menor distancia acumulada para posteriormente unirse a él.

Si se presenta el caso en el que existen dos o más nodos con la misma distancia mínima, la elección del nodo es dependiente de la dirección del barrido de la matriz,

ya que sería el nodo que mejor lo direccione hacia la meta. Este proceso continua hasta haber analizado todos los nodos contenidos en la matriz *M*.

El trazado de la trayectoria se describe en el diagrama de flujo de la Figura 3.11. La ruta es trazada desde el nodo Meta al nodo Inicio; por lo que solo basta con situarse el nodo Meta y verificar cuáles son los nodos de procedencia en la matriz *Dijkstra*. El proceso continua hasta llegar al nodo Inicio. Dado que por cada nodo se analizaron los ocho nodos circundantes al mismo, el algoritmo nos permite trazar rutas ortogonales, cuatro alternativas de movimiento en cada nodo, y omnidireccionales con ocho alternativas por nodo. El resultado se muestra en la Figura 3.12.

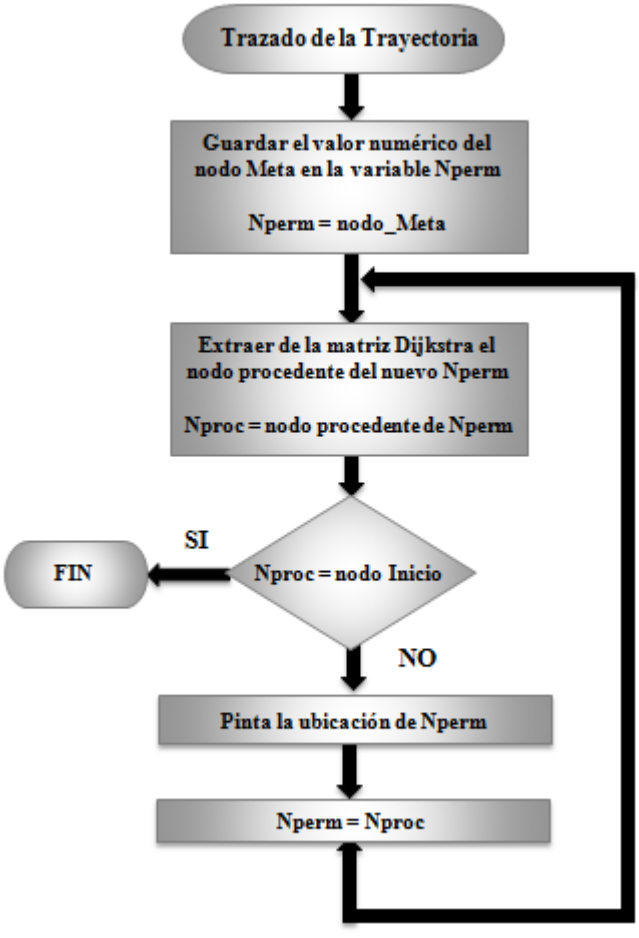
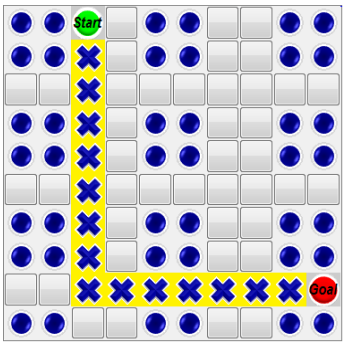
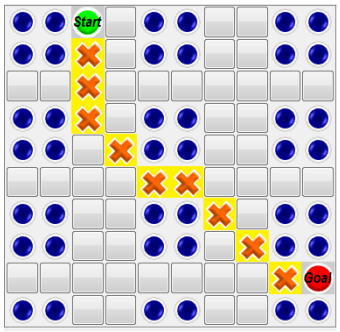


Fig. 3. 11 Diagrama de Flujo del trazado de la trayectoria



(a) Movimientos ortogonales



(b) Movimientos omnidireccionales

Fig. 3. 12 Ruta encontrada por el Algoritmo de Dijkstra

3.4 Comunicación con el robot móvil: Lego NXT

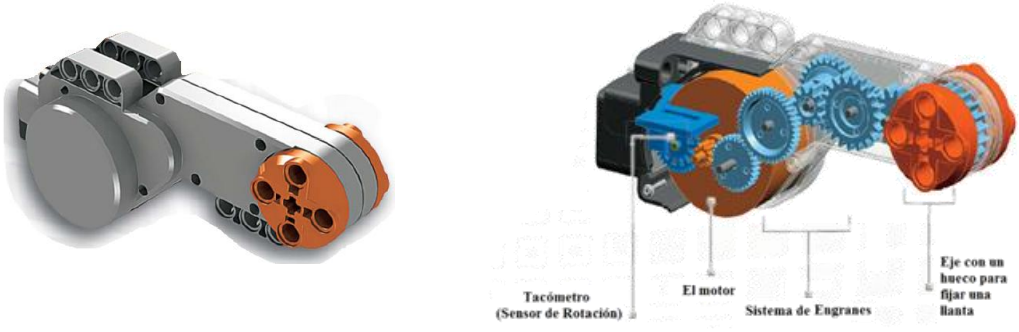
La etapa de comunicación consiste en lograr que el robot Lego NXT realice la ruta libre de colisiones trazada por el algoritmo de Dijkstra. Para realizar tal actividad, es necesaria la traducción de la ruta a instrucciones que el robot pueda interpretar.

Para esta traducción se utiliza la caja de herramientas *"Lego Mindstorms NXT para Matlab y Simulink"* [43]. Dicha caja de herramientas contiene todo un conjunto de funciones escritas en Matlab para ordenarle al robot realizar diferentes tareas, tales como: configuración de sensores, encendido y apagado de motores, lectura de la salida de los sensores, etc. mediante el uso de la interfaz serial Bluetooth.

3.4.1 Calibración de los motores

El robot Lego NXT cuenta con dos motores de corriente continua alimentados con 9 Volts, cada uno tiene un sensor de rotación (encoder) el cual permite tener control sobre los movimientos del robot. El sensor de rotación mide las rotaciones en grados con una precisión de +/- 1°. Una rotación es igual a 360°. La estructura de

cada motor se ilustra en la Figura 3.13, la ubicación del encoder se puede observar en la Figura 3.13-b.



(a) Estructura externa (b) Estructura interna

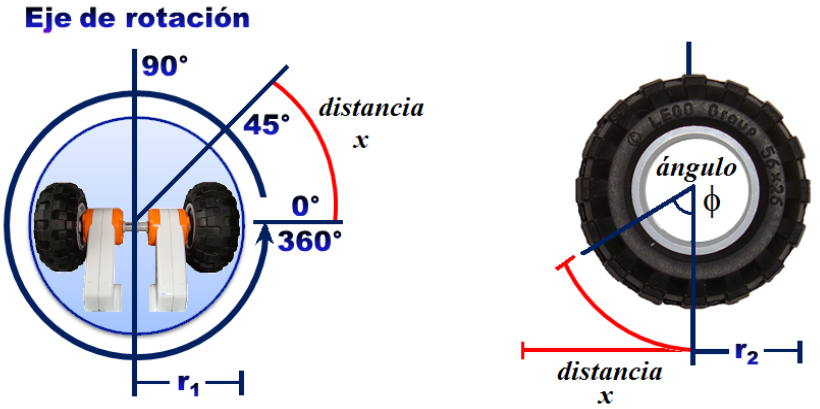
Fig. 3. 13 Estructura de cada motor del Lego NXT

La calibración de los motores consiste en buscar que ambos motores alcancen el mismo número de grados por cada movimiento del robot. Considerando que cada cuadro en el ambiente real es de 20 cm por lado; es preciso establecer que los movimientos en línea recta para avanzar de un cuadro a otro sean por 20 cm y que los giros sean por 45°, 90° y 180°. Para esto, es necesario establecer el número de grados que debe girar cada rueda para que el robot cubra cada uno de los giros y la distancia de 20 cm.

En la Figura 3.14-a se ilustra la base del robot lego NXT, donde r_1 representa el radio de la circunferencia en la que gira el robot sobre su propio eje. Para conocer la distancia que debe recorrer el robot por cada giro se aplica la siguiente fórmula:

$$Perímetro_{arco} = \frac{\pi\, d\, \phi}{360°} \tag{3.8}$$

donde d es el diámetro ($2r$) de la circunferencia y ϕ es el ángulo de giro sobre el perímetro del arco.



(a) Perímetro de rotación de la base del robot

(b) Ángulo de giro sobre la llanta

Fig. 3. 14 Esquema de los ángulos de rotación sobre la base y llanta del robot

La medida del radio de la base del robot es de 4.5 cm; por lo que haciendo uso de la fórmula 3.8 se logra obtener el perímetro de cada uno de los giros. Por ejemplo, para un giro de 45° se obtiene un perímetro de 3.5 cm ilustrado como la *distancia x* en la Figura 3.14-a. Para conocer el ángulo de giro sobre la llanta del robot, basta con despejar la variable ϕ de la fórmula 3.8.

$$\phi = \frac{Perímetro_{arco}\ 360}{\pi\ d} \tag{3.9}$$

La Figura 3.14-b muestra un esquema de la llanta que se encuentra montada sobre el eje de cada motor. El radio ilustrado con la variable r_2 es de 2.85 cm, por lo que aplicando la fórmula 3.9 se obtiene un ángulo de 72°. Este resultado representa que por cada 45° de desplazamiento en el movimiento del robot, el motor debe rotar 72°. Siguiendo con éste mismo procedimiento se obtienen los resultados de la Tabla 3.5.

Tabla 3. 5 Descripción de los ángulos de rotación de cada motor

Desplazamiento por:	Perímetro del arco	Ángulo de rotación del motor
45°	3.5 cm	70°
90°	7 cm	140°
180°	14 cm	281°
20 cm	20 cm	402°

3.4.2 Envío de la ruta al robot móvil

La ruta libre de colisiones entregada por el algoritmo de enrutamiento se envía al robot móvil mediante el uso de la conexión inalámbrica Bluetooth. Para esto, es necesario establecer la conexión entre el robot y la computadora. Los pasos para habilitar dicha conexión se describen en el Apéndice A. Para logar que el robot interprete la ruta libre de colisiones, es necesario convertir la ruta a instrucciones reconocidas por el robot.

Como primer paso, cada movimiento del robot se asocia con una dirección cardinal, las cuales están descritas en la Tabla 3.6. De esta manera, la trayectoria total se traduce como una serie de direcciones cardinales. El pseudocódigo de esta traducción se detalla en la Tabla 3.7.

Tabla 3. 6 Direcciones correspondientes a cada movimiento

Movimiento hacia:	Dirección	Movimiento hacia:	Dirección
Arriba	Norte	Abajo	Sur
Arriba y Derecha	Noreste	Abajo e Izquierda	Suroeste
Derecha	Este	Izquierda	Oeste
Derecha y Abajo	Sureste	Izquierda y Arriba	Noroeste

La trayectoria entregada por el algoritmo 3.7 sirve para determinar la dirección en la que debe moverse el robot para avanzar de nodo a nodo hasta completar la ruta. Sin embargo, es necesaria una segunda traducción para indicarle al robot el movimiento en grados que debe realizar para encaminarse a cada dirección marcada por la trayectoria. Estos movimientos son por ± 45°, ±90° y 180°. Cabe mencionar que cuando el robot se encuentra alineado sobre la dirección establecida, se envía una orden para que avance en línea recta por 20cm.

Tabla 3. 7 Pseudocódigo del algoritmo de traducción de la ruta

Algoritmo 3.7. Traducción a direcciones cardinales

```
Entrada: Los nodos Inicio y Meta y la matriz Grafo
Salida: La ruta descrita en direcciones cardinales
1:  Nperm = Nodo Meta
2:  Trayectoria = [ ]
2:  while Nperm ≠ NStart
3:      Nproc = Nodo procedente de Nperm
4:      if Nproc = Grafo (Nperm). Arriba
5:          Dirección = Norte
6:      elseif Nproc = Grafo (Nperm). Derecha
7:          Dirección = Este
8:      elseif Nproc = Grafo (Nperm). Abajo
9:          Dirección = Sur
11:     elseif Nproc = Grafo (Nperm). Izquierda
12:         Dirección = Oeste
13:     elseif Nproc = Grafo (Nperm). ArribaDerecha
14:         Dirección = Noreste
15:     elseif Nproc = Grafo (Nperm). AbajoDerecha
16:         Dirección = Sureste
17:     elseif Nproc = Grafo (Nperm). ArribaIzquierda
18:         Dirección = Noroeste
19:     elseif Nproc = Grafo (Nperm). AbajoIzquierda
20:         Dirección = Suroeste
21:     end if
22:   Trayectoria = Dirección ∪ Trayectoria
23:   Nperm = Nproc
24:  end while
```

Como condición de inicio, se establece que el robot debe estar apuntando hacia la dirección Norte, tal como se muestra en la Figura 3.15. La Tabla 3.8 describe los once movimientos que debe realizar el robot para completar la ruta de la Figura 3.16. Los giros positivos son en sentido contrario a las manecillas del reloj y los negativos en el sentido contrario.

Mediante el uso de las funciones contenidas en la caja de herramientas *"Lego Mindstorms NXT para Matlab y Simulink"* se implementaron seis rutinas básicas para describir cada movimiento. Los movimientos que involucren rotaciones por ±130° se efectúan por medio de dos giros: uno por ±90° y otro por ±45°.

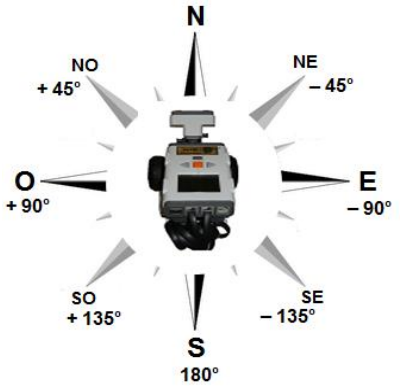


Fig. 3. 15 Esquema de la posición inicial del robot

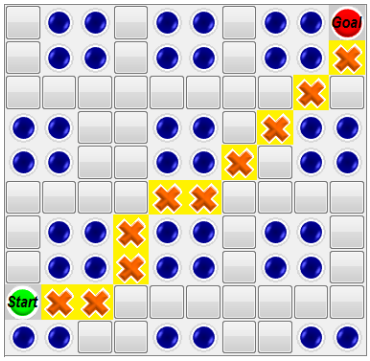


Fig. 3. 16 Esquema de una ruta entregada por el algoritmo de Dijkstra

Tabla 3. 8 Movimientos para efectuar la trayectoria 3.15

	Trayectoria descrita en direcciones cardinales	Movimientos del robot
1:	Este	Girar -45° y Avanzar 20 cm
2:	Este	Avanzar 20 cm
3:	Noreste	Girar +45° y Avanzar 20 cm
4:	Norte	Girar +45° y Avanzar 20 cm
5:	Noreste	Girar -45° y Avanzar 20 cm
6:	Este	Girar -45° y Avanzar 20 cm
7:	Noreste	Girar +45° y Avanzar 20 cm
8:	Noreste	Avanzar 20 cm
9:	Noreste	Avanzar 20 cm
10:	Noreste	Avanzar 20 cm
11:	Norte	Girar +45° y Avanzar 20 cm

En cada rutina, la velocidad en ambos motores es siempre la misma. Sin embargo; por cada giro, la dirección de rotación entre cada motor es opuesta. En la Tabla 3.9 se detallan los movimientos de los motores para encaminarse a cada posible dirección cuando el robot se encuentra apuntando hacia el Norte.

Tabla 3. 9 Descripción de los movimientos del robot en la condición inicial

Hacia la dirección:	Movimiento del robot	Ángulo de rotación de los motores	Movimiento del Motor Derecho (MD) y del Motor Izquierdo (MI)
Norte	20 cm en línea recta	MD = + 402° MI = + 402°	M D ← Línea recta M I ← Línea recta
Noreste	– 45°	MD = – 70° MI = + 70°	MD← Avanza hacia atrás M I ← Avanza hacia adelante
Este	–90°	MD = – 140° MI = + 140°	MD ← Avanza hacia atrás M I ← Avanza hacia adelante
Sureste	–135° = –90 –45°	MD = –140° – 70 MI = + 140° + 70°	MD ← Avanza hacia atrás M I ← Avanza hacia adelante
Sur	180°	MD = + 280° MI = –280°	MD ← Avanza hacia adelante M I ← Avanza hacia atrás
Suroeste	135° = +90° + 45°	MD = +140° + 70° MI = –140° –70°	MD ← Avanza hacia adelante M I ← Avanza hacia atrás
Oeste	90°	MD = + 140° MI = – 140°	MD ← Avanza hacia adelante M I ← Avanza hacia atrás
Noroeste	45°	MD = +70° MI = - 70°	MD ← Avanza hacia adelante M I ← Avanza hacia atrás

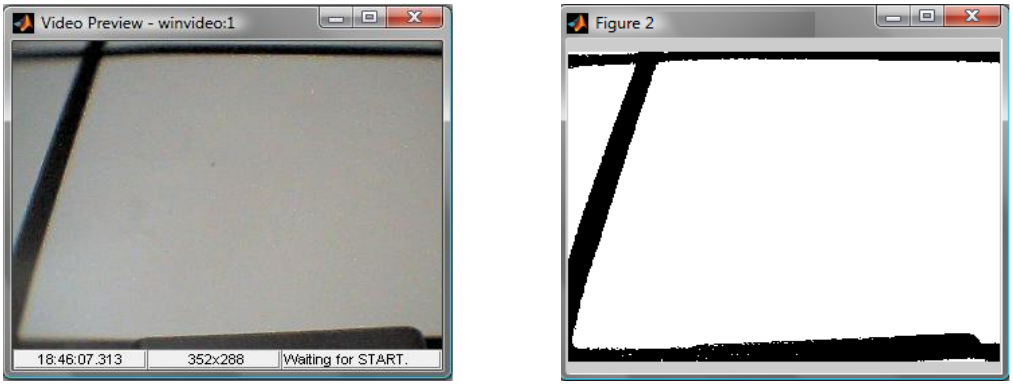
3.5 Ajuste de verticalidad por Visión y Control Difuso

Cuando el robot Lego se desplaza de un nodo hacia otro, existe la posibilidad de que su ruta se desvíe ligeramente; especialmente en los giros de 90°. Por esta razón es necesario incluir la etapa de corrección de ruta en la forma de un control de lazo cerrado con retroalimentación en base a información visual. Esta etapa consiste en el monitoreo constante de la línea de la acotación del camino. Si esta última es detectada quiere decir que el robot ha realizado una desviación indeseable.

La lógica difusa y el procesamiento de imágenes han demostrado resultados favorables en aplicaciones de detección y seguimiento de algún objeto [5]. La información visual requerida por el sistema de control difuso es captada a través de una web-cam e importada hacia el entorno de trabajo de MATLAB por medio de sus funciones de adquisición de imágenes.

Con el objeto de obtener información numérica acerca de la posición de la línea de acotamiento detectada por la cámara, se lleva a cabo una serie de operaciones de pre-procesamiento sobre la señal de video de entrada. La cámara entrega originalmente una secuencia de imágenes a color con una resolución de 352x288.

La imagen de entrada es binarizada con un umbral obtenido a través de las funciones de procesamiento de imágenes de Matlab. La figura 3.17 muestra las imágenes antes y después del pre-procesamiento.



(a) Imagen real (b) Imagen binarizada

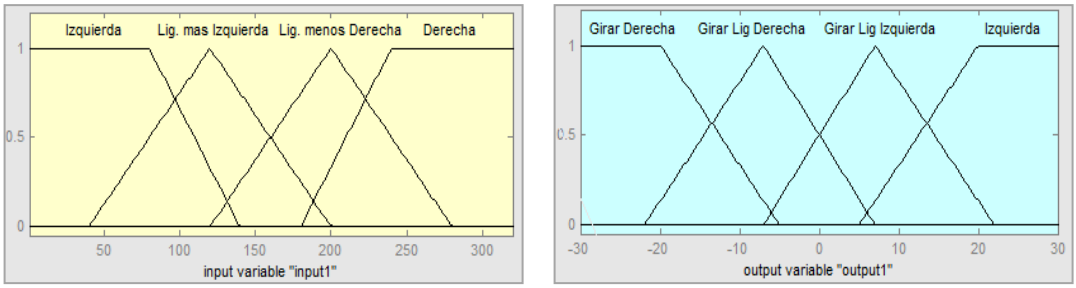
Fig. 3. 17 Captura de la acotación del camino por la web-cam montada

El control difuso requerido para éste sistema es a una sola entrada y una sola salida. Éste consiste en tomar como dato de entrada, la información numérica correspondiente a la posición de la línea de acotamiento detectada por la cámara web, para posteriormente entregar como salida un nuevo dato numérico correspondiente al desplazamiento en grados en el movimiento del robot.

Las funciones de membresía introducidas a la entrada y salida del control difuso son: dos de tipo “trapezoidal” y dos de tipo “triangular”. Dichas funciones de membresía se describen en la Tabla 3.10 y su representación gráfica se muestra en la Figura 3.18.

Tabla 3. 10 Funciones de Membresía del Controlador Difuso

FUNCIONES DE MEMBRESÍA DE ENTRADA		
Variable	**Tipo**	**Partición (Pixeles)**
Izquierda	Trapezoidal	[1 1 90 150]
Ligeramente más a la Izquierda	Triangular	[40 120 200]
Ligeramente menos a la Derecha	Triangular	[120 200 280]
Derecha	Trapezoidal	[220 280 352 352]
FUNCIONES DE MEMBRESÍA DE SALIDA		
Variable	**Tipo**	**Partición (Grados)**
Girar a la Derecha	Trapezoidal	[-30 -30 -20 -5]
Girar Ligeramente a la Derecha	Triangular	[-22 -7 7]
Girar Ligeramente a la Izquierda	Triangular	[-7 7 22]
Girar a la Izquierda	Trapezoidal	[5 20 30 30]



(b) Entrada

(a) Salida

Fig. 3. 18 Esquemas de las Funciones de Membresía introducidas al controlador difuso

Las reglas de inferencia para el control son basadas en las reglas heurísticas IF (antecedente) THEN (consecuente). En un sistema de control difuso, en el cual las variables de entrada tienen el mismo número de funciones de membresía, el número de reglas está dado por [5]:

$$r = n^i \tag{3.10}$$

donde: r es el número total de reglas, n es el número de conjuntos difusos en la partición de las variables de entrada e i es el número total de entradas.

De acuerdo con la ecuación 3.10, para el sistema se implementaron cuatro reglas de inferencia. Dado que se requiere que el desplazamiento del robot sea en dirección contraria a la posición de la línea de acotamiento detectada, las reglas son propuestas de tal manera que se cumpla con lo anterior. Las cuatro reglas de inferencia se describen en la Tabla 3.11.

Tabla 3. 11 Reglas de Inferencia para el Controlador Difuso

Regla	Entrada (IF)	Salida (THEN)
1	Izquierda	Girar Ligeramente a la Derecha
2	Ligeramente más a la Izquierda	Girar a la Derecha
3	Ligeramente menos a la Derecha	Girar a la Izquierda
4	Derecha	Girar Ligeramente a la Izquierda

3.6 Operación automática del sistema por visión y ultrasonido

Con la finalidad de construir un algoritmo en el cual el robot sea capaz de explorar el entorno, buscar las colisiones presentes y reportar la localización de los mismos. Se implementó una rutina con el uso de un sensor de ultrasonido y una web-cam, ambos montados sobre el robot.

En esta rutina, el sensor de ultrasonido le permite al robot detectar los obstáculos presentes en su entorno y medir la distancia a la que se encuentra de ellos. Según las especificaciones del sensor se pueden detectar objetos situados a distancias entre los 0 y 255 cm, con una precisión de ± 3 cm.

Tal como se establece en el diagrama de operación de la Figura 3.3, el usuario establece únicamente la posición de los nodos INICIO y META. Posteriormente, se ejecuta el algoritmo de Dijkstra como si no existieran obstáculos presentes en el ambiente. Si el sensor de ultrasonido detecta un obstáculo a una distancia menor de

20 cm, el robot se detiene para retroalimentarle a la computadora la posición de dicho obstáculo.

Si la posición del obstáculo interfiere con la ruta trazada por el algoritmo de Dijkstra, la ubicación actual del robot se establece como un nuevo nodo de Inicio y el algoritmo de enrutamiento es nuevamente ejecutado. De lo contrario; si algún obstáculo es detectado por el sistema, pero no interfiere en la ruta simplemente se retroalimenta la posición y el robot continua con su recorrido. Este proceso continua hasta que el robot alcance llegar a la meta establecida inicialmente.

Fig. 3. 19 Sistema de desplazamiento con localización y evasión de obstáculos

De igual manera, si se detectan desviaciones indeseables por parte del robot se ejecuta la etapa de corrección de verticalidad descrita en la sección 3.5. La Figura 3.19 presenta el esquema de este modo de operación.

En éste capítulo se ha descrito el procedimiento de diseño e implementación de cada una de las etapas involucradas en el funcionamiento del sistema. Se mostraron en forma de pseudocódigo los diferentes algoritmos realizados en Matlab. En el siguiente capítulo se detallarán los resultados de la implementación del sistema de navegación en sus diferentes modos de funcionamiento.

Capítulo 4

Experimentos y resultados

En este capítulo se presentan los experimentos realizados para comprobar el funcionamiento del sistema de navegación implementado en Matlab. De igual manera se muestra la caracterización del sensor de ultrasonido montado en el robot móvil y la validación de las rutinas de captura del ambiente real y de ajuste de verticalidad en el recorrido del robot por control difuso.

4.1 Caracterización del sensor de ultrasonido

El sensor de ultrasonido utilizado en el sistema es el que viene incluido en el Kit Mindstorm NXT. El fabricante especifica que el sensor es capaz de medir distancias entre 0 y 2.5 m y la lectura puede ser entregada en centímetros y pulgadas. De igual manera, éste especifica que los objetos planos proporcionan mejores lecturas y aquellos que son muy pequeños, delgados o en forma curva pueden presentar dificultades.

Dado que el fabricante no especifica la apertura del haz de la onda sonora, es necesario realizar algunas pruebas para conocer si los objetos que no se encuentren estrictamente frente al sensor interfieren en la lectura. De igual manera, es esencial comprobar la distancia mínima real en la que un objeto es detectado; ya que la aplicación exige que el sensor detecte obstáculos a una distancia de al menos 20 cm.

El experimento para caracterizar el sensor de ultrasonido consiste colocar un obstáculo a diferentes distancias y ángulos con respecto al sensor, tal como se muestra en la Figura 4.1 y obtener la lectura entregada por este último. La máxima distancia con la que se trabajó en este experimento fue por 50 cm y el obstáculo es un prisma rectangular de base 6x13 cm y altura de 13 cm.

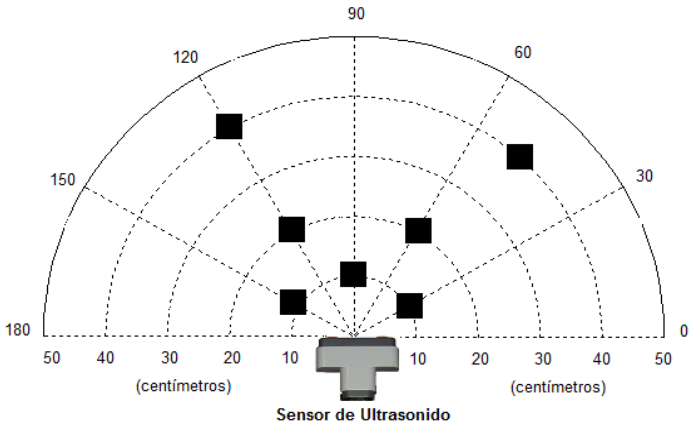
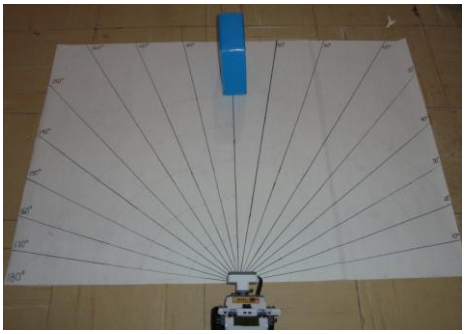


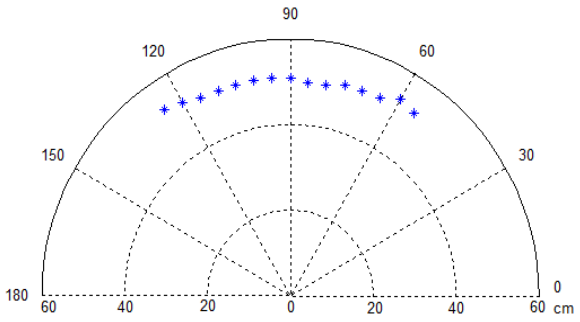
Fig. 4. 1 Esquema de la caracterización del sensor de ultrasonido

Las Figuras de la 4.2 a la 4.6 ilustran los ejemplos de los experimentos realizados para caracterizar al sensor y las Tablas de la 4.1 a la 4.5 detallan los resultados obtenidos. El error presentado en cada experimento fue calculado con la siguiente fórmula:

$$\%\, Error\ = \left| \frac{Distancia\ real - Distancia\ medida}{Distancia\ real} \right| x100 \qquad (4.1)$$



(a) Detección del objeto a 50 cm de distancia

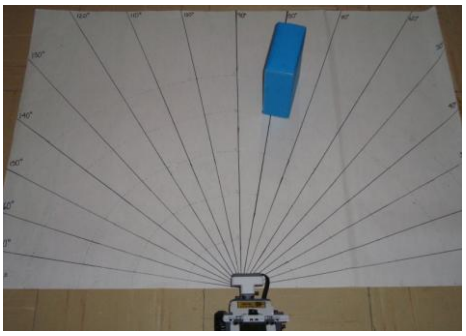


(b) Distancias entregadas por el sensor

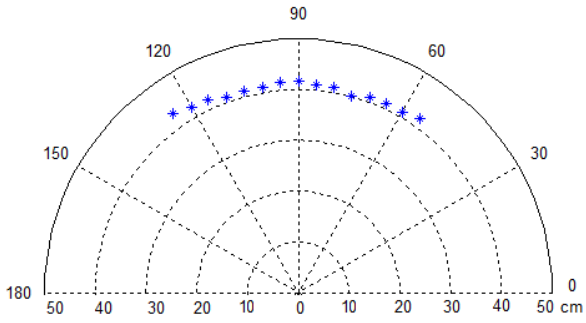
Fig. 4. 2 Caracterización del sensor con objetos colocados a 50 cm de distancia

Tabla 4. 1 Resultados obtenidos en la detección de obstáculos a 50 cm (± 1 cm) de distancia

Ángulo (±1°)	Lectura del sensor (cm)	Error (%)	Ángulo (±1°)	Lectura del sensor (cm)	Error (%)
55°	52 cm	4 %	**95°**	51 cm	2 %
60°	53 cm	6 %	**100°**	51 cm	2 %
65°	51 cm	2 %	**105°**	51 cm	2 %
70°	51 cm	2 %	**110°**	51 cm	2 %
75°	51 cm	2 %	**115°**	51 cm	2 %
80°	50 cm	0 %	**120°**	52 cm	4 %
85°	50 cm	0 %	**125°**	53 cm	6 %
90°	51 cm	2 %			



(a) Detección del objeto a 40 cm de distancia

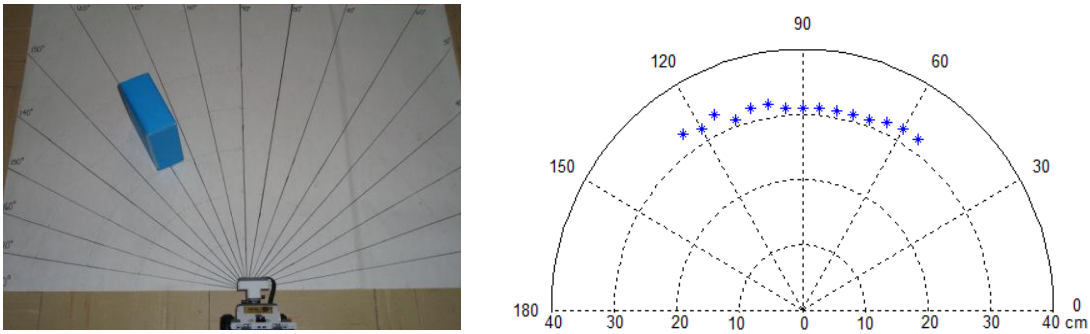


(b) Distancias entregadas por el sensor

Fig. 4. 3 Caracterización del sensor con objetos colocados a 40 cm de distancia

Tabla 4. 2 Resultados obtenidos en la detección de obstáculos a 40 cm (± 1 cm) de distancia

Ángulo (±1°)	Lectura del sensor (cm)	Error (%)	Ángulo (±1°)	Lectura del sensor (cm)	Error (%)
55°	41.7 cm	4.25 %	**95°**	41.6 cm	4 %
60°	41 cm	2.5 %	**100°**	41 cm	2.5 %
65°	41 cm	2.5 %	**105°**	41 cm	2.5 %
70°	41 cm	2.5 %	**110°**	41 cm	2.5 %
75°	41 cm	2.5 %	**115°**	42 cm	5 %
80°	41 cm	2.5 %	**120°**	42 cm	5 %
85°	41 cm	2.5 %	**125°**	43 cm	7.5 %
90°	41.6 cm	4 %			

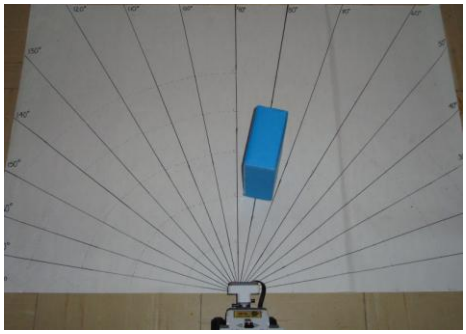


(a) Detección del objeto a 30 cm de distancia (b) Distancias entregadas por el sensor

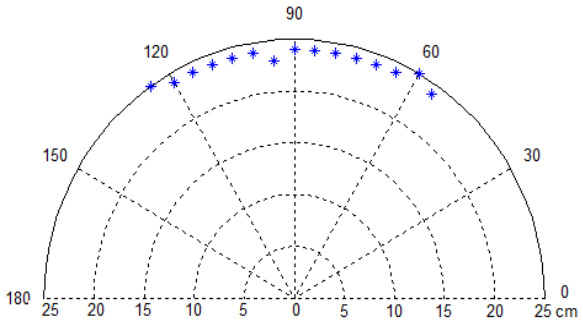
Fig. 4. 4 Caracterización del sensor con objetos colocados a 30 cm de distancia

Tabla 4. 3 Resultados obtenidos en la detección de obstáculos a 30 cm (± 1 cm) de distancia

Ángulo (±1°)	Lectura del sensor (cm)	Error (%)	Ángulo (±1°)	Lectura del sensor (cm)	Error (%)
55°	32 cm	6.66 %	**95°**	31 cm	3.33 %
60°	32 cm	6.66 %	**100°**	32 cm	6.66 %
65°	31.8 cm	6 %	**105°**	32 cm	6.66 %
70°	31 cm	3.33 %	**110°**	31 cm	3.33 %
75°	31 cm	3.33 %	**115°**	33 cm	7.5 %
80°	31 cm	3.33 %	**120°**	32 cm	6.66 %
85°	31 cm	3.33 %	**125°**	33 cm	7.5 %
90°	31 cm	3.33 %			



(a) Detección del objeto a 20 cm de distancia

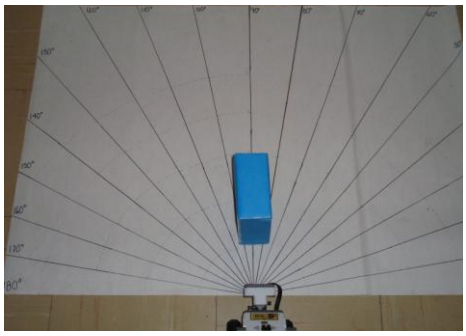


(b) Distancias entregadas por el sensor

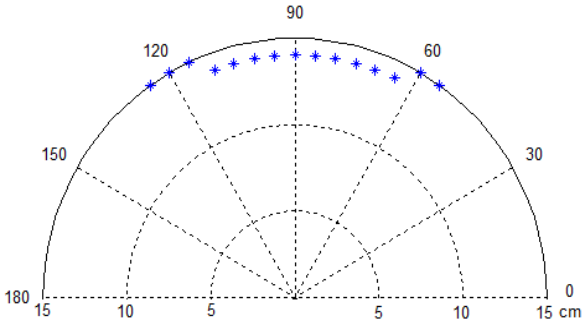
Fig. 4. 5 Caracterización del sensor con objetos colocados a 20 cm de distancia

Tabla 4. 4 Resultados obtenidos en la detección de obstáculos a 20 cm (± 1 cm) de distancia

Ángulo (±1°)	Lectura del sensor (cm)	Error (%)	Ángulo (±1°)	Lectura del sensor (cm)	Error (%)
55°	24 cm	20 %	**95°**	23 cm	15 %
60°	25 cm	25 %	**100°**	24 cm	20 %
65°	24 cm	20 %	**105°**	24 cm	20 %
70°	24 cm	20 %	**110°**	24 cm	20 %
75°	24 cm	20 %	**115°**	24.10 cm	20.5 %
80°	24 cm	20 %	**120°**	24 cm	20 %
85°	24 cm	20 %	**125°**	25 cm	25 %
90°	24 cm	20 %			



(a) Detección del objeto a 10 cm de distancia



(b) Distancias entregadas por el sensor

Fig. 4. 6 Caracterización del sensor con objetos colocados a 10 cm de distancia

Tabla 4. 5 Resultados obtenidos en la detección de obstáculos a 10 cm (± 1 cm) de distancia

Ángulo (±1°)	Lectura del sensor (cm)	Error (%)	Ángulo (±1°)	Lectura del sensor (cm)	Error (%)
55°	15 cm	50 %	**95°**	14 cm	40 %
60°	15 cm	55 %	**100°**	14 cm	40 %
65°	15 cm	50 %	**105°**	14 cm	40 %
70°	14 cm	40 %	**110°**	14 cm	40 %
75°	14 cm	40 %	**115°**	14 cm	40 %
80°	14 cm	40 %	**120°**	15 cm	50 %
85°	14 cm	40 %	**125°**	15 cm	50 %
90°	14 cm	40 %			

De igual manera se colocó el objeto a distancias menores de 10 cm y se comprobó que la mínima lectura entregada por el sensor es de 6 cm; es decir, si un objeto se encuentra a 1 cm de distancia con respecto al sensor, éste entrega 6 cm como lectura.

De acuerdo con los experimentos anteriores, se concluye que el rango de apertura del sensor ultrasónico se encuentra entre los 55° y 125°. Así como también se observó que a distancias menores ó iguales a los 20 cm, las lecturas del sensor presentan un error mayor ó igual al 20%, el cual es mucho mayor con respecto al error que se presenta con objetos colocados a mayores distancias, como por ejemplo a 30 cm. Sin embargo, se puede concluir que el comportamiento del sensor es muy estable, ya que el aumento en el error puede verse como la presencia de un offset interno del sensor que se hace evidente a menores distancias.

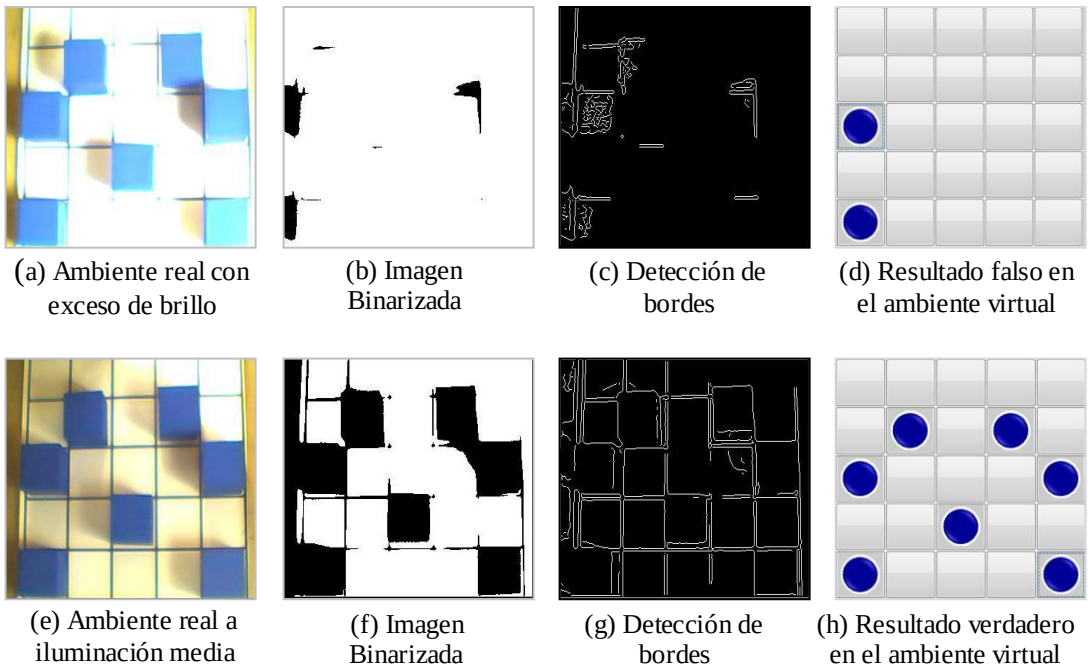
4.2 Validación de la captura automática de obstáculos

Para comprobar el buen funcionamiento del algoritmo de captura automática de obstáculos, se construyó físicamente un ambiente real de 25 cuadros de longitud 20x20 cm. Una cámara web con resolución de 352x288 pixeles se encuentra a una altura de 2.25 m. apuntando directamente hacia el entorno real sobre el cual se desplaza el robot.

Las condiciones de iluminación son muy importantes en el desempeño eficiente de esta etapa; por lo que se realizaron las pruebas con diversas disposiciones de obstáculos bajo distintas condiciones iluminación.

El exceso de iluminación durante las pruebas de día provoca que uno ó varios obstáculos sean eliminados en el proceso de binarización, por lo que se reportan posiciones libres en nodos donde en realidad se encuentran obstáculos. Por otra parte, las sombras que se generan, provocan que el algoritmo establezca obstáculos sobre las posiciones libres.

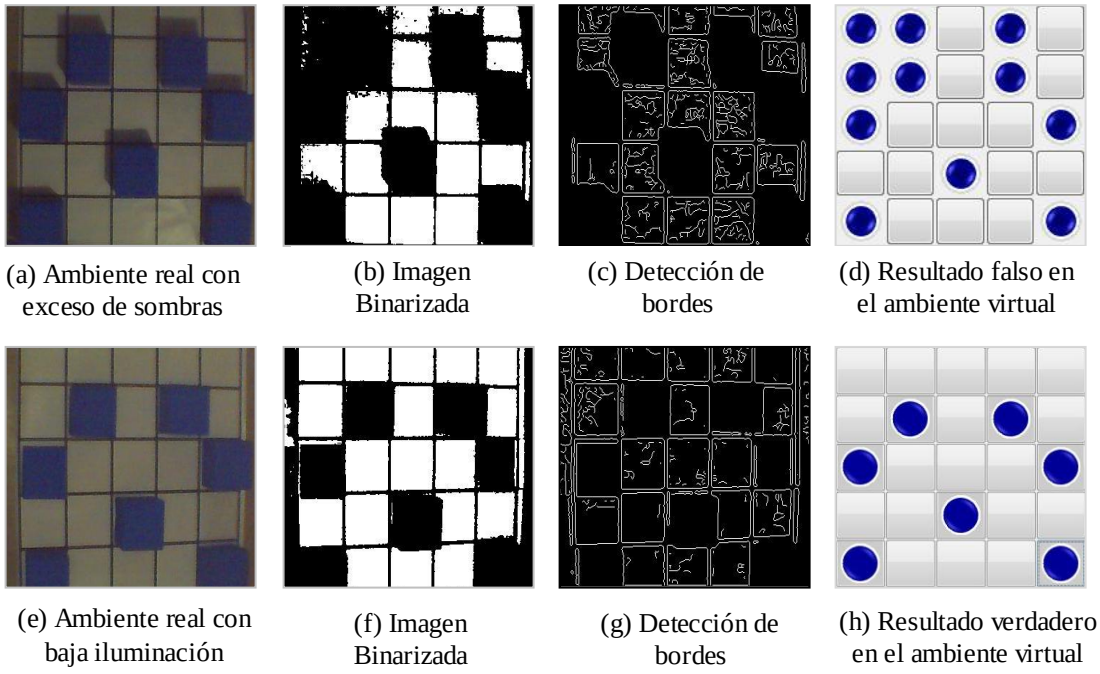
Las Figuras 4.7 y 4.8 muestran ejemplos de la ejecución de esta prueba y de los problemas que se presentan. La Tabla 4.6 resume los resultados obtenidos.



(a) Ambiente real con exceso de brillo | (b) Imagen Binarizada | (c) Detección de bordes | (d) Resultado falso en el ambiente virtual

(e) Ambiente real a iluminación media | (f) Imagen Binarizada | (g) Detección de bordes | (h) Resultado verdadero en el ambiente virtual

Fig. 4. 7 Captura del ambiente real bajo iluminación natural del día



(a) Ambiente real con exceso de sombras | (b) Imagen Binarizada | (c) Detección de bordes | (d) Resultado falso en el ambiente virtual

(e) Ambiente real con baja iluminación | (f) Imagen Binarizada | (g) Detección de bordes | (h) Resultado verdadero en el ambiente virtual

Fig. 4. 8 Captura del ambiente real bajo iluminación artificial por la noche

Tabla 4. 6 Resultados de la Captura Automática de los Obstáculos por Visión

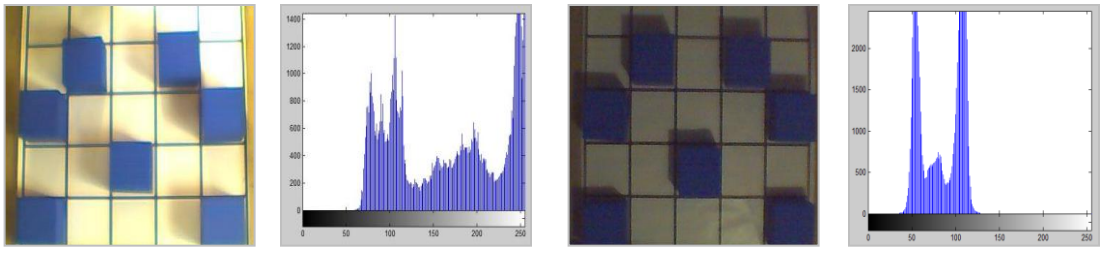
No. De Obstáculos	Total de Pruebas	No. de Obstáculos detectados	No. de Obstáculos fallados	No. de eventos presentados	Porcentaje de éxito
3	10	3	0	6	60%
		2	1 (-)	3	
		4	1 (+)	1	
4	10	3	1 (-)	1	70%
		4	0	7	
		5	1 (+)	1	
		6	2 (+)	1	
5	10	3	2 (-)	1	70%
		5	0	7	
		7	1 (+)	2	
6	10	4	2 (-)	1	80%
		6	0	8	
		8	2 (+)	1	
7	10	2	5 (-)	1	70%
		7	0	7	
		9	2 (+)	1	
		10	3 (+)	1	

De acuerdo con los resultados expuestos en la Tabla 4.6, no es posible validar como eficiente el desempeño de esta etapa; por lo que es necesario analizar las imágenes para determinar las características de aquellas que proporcionaron resultados exitosos.

Con el objetivo de obtener un criterio de clasificación que sea acorde con la iluminación presente en las imágenes se establece un factor de contraste. Este consiste en determinar la razón entre el número de pixeles que se encuentran por arriba (P_{sup}) y por debajo (P_{inf}) de un umbral, el cual es el valor medio en el rango de nivel de grises (de 0 a 256). Los valores P_{sup} y P_{inf} son extraídos del histograma de la imagen.

$$Factor\ de\ contraste = \frac{P_{sup}}{P_{inf}} \tag{4.2}$$

La Figura 4.9 muestra el histograma de dos imágenes que reportan resultados exitosos y la Tabla 4.7 muestra el factor de contraste de las 15 imágenes más representativas del total de las 50 imágenes capturas.



(a) Imagen con factor de contraste de 1.697 (b) Imagen con factor de contraste de 0.0065

Fig. 4. 9 Histogramas de imágenes con diferente factor de contraste

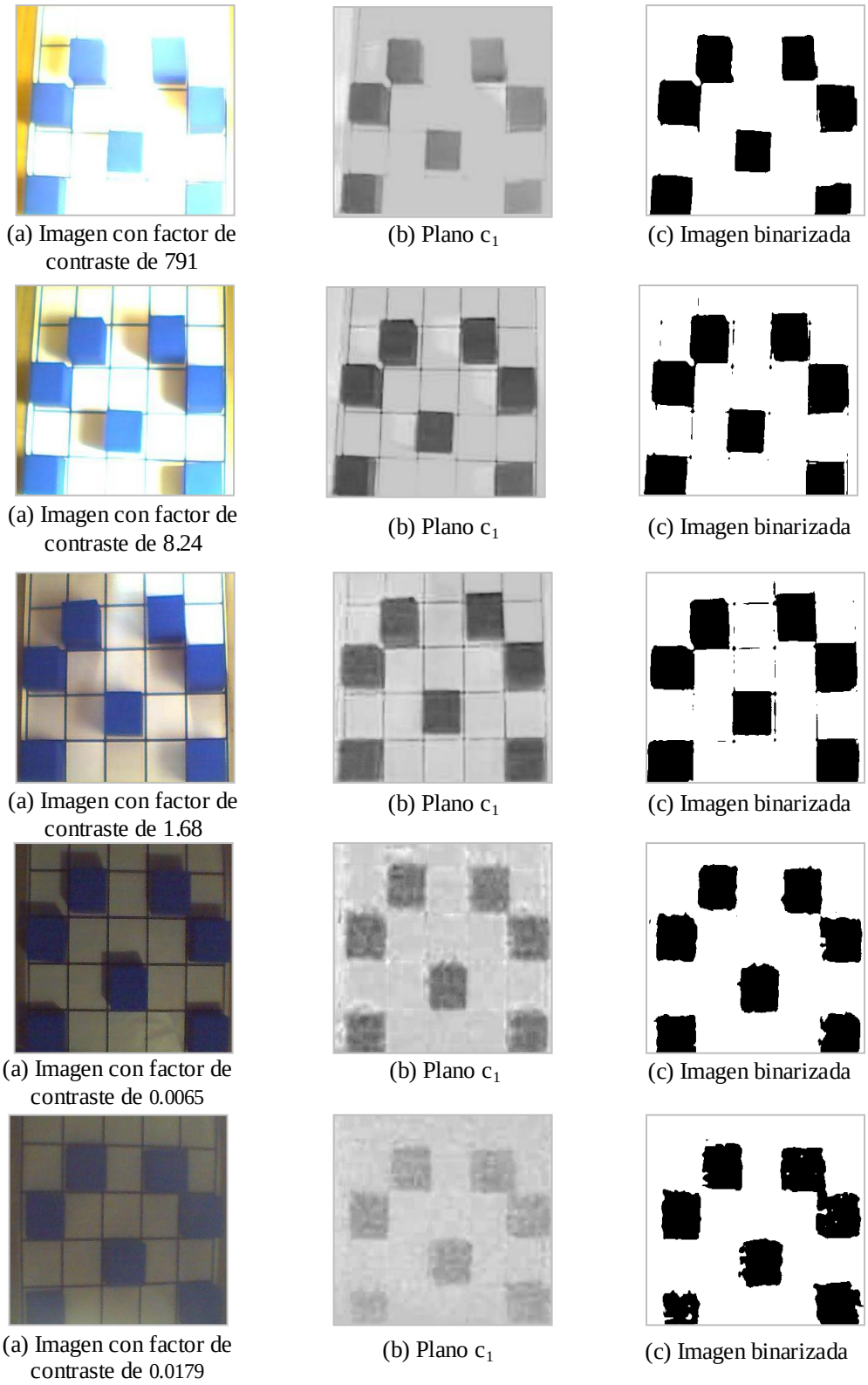
Tabla 4. 7 Factor de Contraste en las imágenes más representativas

Caso		Psup	Pinf	Factor de Contraste
1	Mal Resultado	101248	128	791
2	Mal Resultado	98534	2842	34.67
3	Mal resultado	98095	3281	29.89
4	Mal resultado	95543	5833	16.37
5	Mal resultado	90416	10960	8.24
6	Buen resultado	89734	11642	7.70
7	Buen resultado	87020	14356	6.06
8	Buen resultado	63566	37810	1.68
9	Buen resultado	56633	44743	1.26
10	Mal resultado	654	100722	0.0065
11	Mal resultado	714	100662	0.0071
12	Mal resultado	1784	99592	0.0179
13	Mal resultado	92	101284	9.08×10^{-4}
14	Mal resultado	85	101291	8.391×10^{-4}
15	Mal resultado	37	101339	3.651×10^{-4}

De acuerdo con los resultados expuestos en la Tabla 4.7, se puede concluir que el algoritmo reporta resultados exitosos para aquellas imágenes cuyo factor de contraste se encuentre entre el rango de 1.3 a 7.5.

Debido a la necesidad de ampliar el rango de imágenes que reporten resultados exitosos es necesario que el algoritmo sea más tolerante a las sombras y a los cambios en la iluminación. Para ello se decidió trabajar con el modelo de color c_1, c_2 y c_3 y se observó que el plano c_1 es el que mejor elimina las sombras y resalta los obstáculos.

Tal como se puede apreciar en la Figura 4.10, con la binarización del plano c_1 todos los obstáculos son detectados, siendo el algoritmo más tolerante las condiciones de iluminación y las sombras. Se repitieron las pruebas descritas en la Tabla 4.6, las cuales abarcan un factor de contraste desde 3.61×10^{-4} a 791 y se un obtuvo un 100% de éxito en todas ellas; por lo que se valida como satisfactorio el desempeño de esta etapa.



(a) Imagen con factor de contraste de 791 (b) Plano c_1 (c) Imagen binarizada

(a) Imagen con factor de contraste de 8.24 (b) Plano c_1 (c) Imagen binarizada

(a) Imagen con factor de contraste de 1.68 (b) Plano c_1 (c) Imagen binarizada

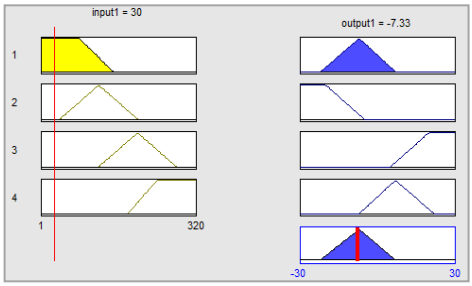
(a) Imagen con factor de contraste de 0.0065 (b) Plano c_1 (c) Imagen binarizada

(a) Imagen con factor de contraste de 0.0179 (b) Plano c_1 (c) Imagen binarizada

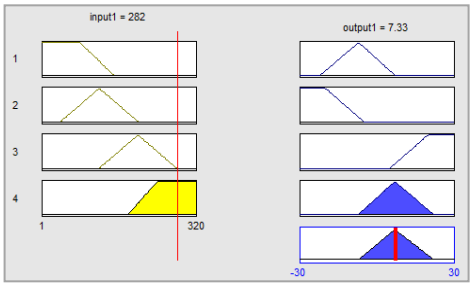
Fig. 4. 10 Resultados del algoritmo tolerante a los cambios en la iluminación

4.3 Validación del controlador difuso

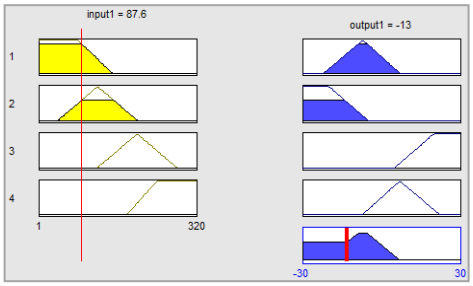
Con la finalidad de comprobar el buen funcionamiento del control difuso, se realiza una simulación haciendo uso de la herramienta gráfica del Sistema de Inferencia Difuso incluido en la caja de herramientas de lógica difusa de Matlab. Se obtienen los gráficos de la Figura 4.11, en la cual se observa el comportamiento de las cuatro reglas del controlador sobre las funciones de membresía.



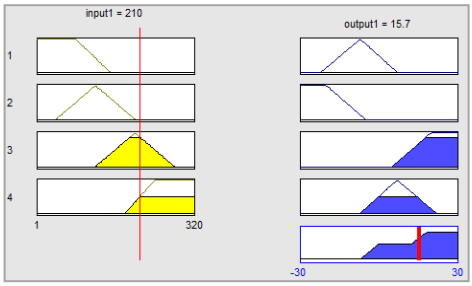
(a) Entrada = pixel 30 (Izquierda), Salida = -7.33° (Girar ligeramente a la Derecha)



(b) Entrada = pixel 282 (Derecha), Salida = 7.33° (Girar ligeramente a la Izquierda)



(c) Entrada = pixel 87.6 (Ligeramente más a la Izquierda), Salida = -13° (Girar a la Derecha)



(d) Entrada = pixel 210 (Ligeramente menos a la Derecha), Salida = 15.7° (Girar a la Izquierda)

Fig. 4. 11 Simulación de las reglas de inferencia en el editor difuso de Matlab

Tabla 4. 8 Resultados del Controlador Difuso a diferentes entradas

Entrada	Salida	Girar:	Entrada	Salida	Girar:
10	-7.33°	Ligeramente a la Derecha	200	17.2°	A la Izquierda
23	-7.33°	Ligeramente a la Derecha	228	13.5°	A la Izquierda
56.4	-9.54°	Ligeramente a la Derecha	251	11.1°	A la Izquierda
101	-14.5°	A la Derecha	268	8.89°	Ligeramente a la Izquierda
121	-17°	A la Derecha	294	7.33°	Ligeramente a la Izquierda

De acuerdo con los resultados expuestos en la Tabla 4.8, se observa que el controlador difuso cumple con las reglas de inferencia expuestas en la Tabla 3.11.

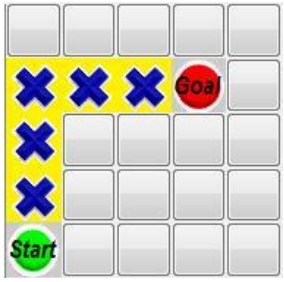
4.4 Comportamiento del robot durante el recorrido de la trayectoria

Para conocer las capacidades de navegación del robot dentro de su entorno, se llevaron a cabo diferentes experimentos para evaluar los comportamientos propuestos para el sistema. Con fines de documentación de éste trabajo, se presentan tres experimentos para mostrar el comportamiento del robot en cada caso.

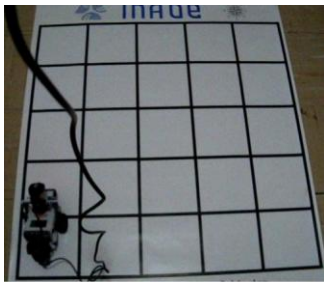
4.4.1 Recorrido sin ajuste de verticalidad por visión y control difuso

La primera prueba consiste en hacer recorrer al robot la trayectoria entregada por el algoritmo de Dijkstra sin la rutina de corrección de verticalidad y de esta manera visualizar qué tanto se desvía el robot de la ruta.

Debido a que se espera que el robot desvíe el recorrido y para evitar colisiones, no se incluyen obstáculos en el entorno. La Figura 4.12 ilustra el recorrido real del robot durante el experimento y la Figura 4.13 es una gráfica comparativa entre la trayectoria ideal y la trayectoria real.



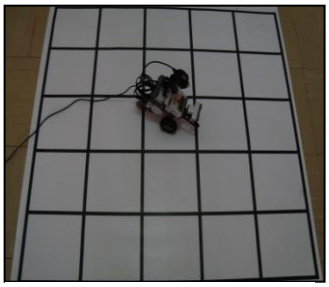
(a) Ruta entregada por el algoritmo de Dijkstra



(b) Posición inicial del robot

(c) Primer desvío en el giro de 90° a la derecha

(d) El robot avanza desviado



(e) El robot avanza desviado

(f) Posición final del robot

Fig. 4. 12 Comportamiento del robot en el recorrido de la trayectoria

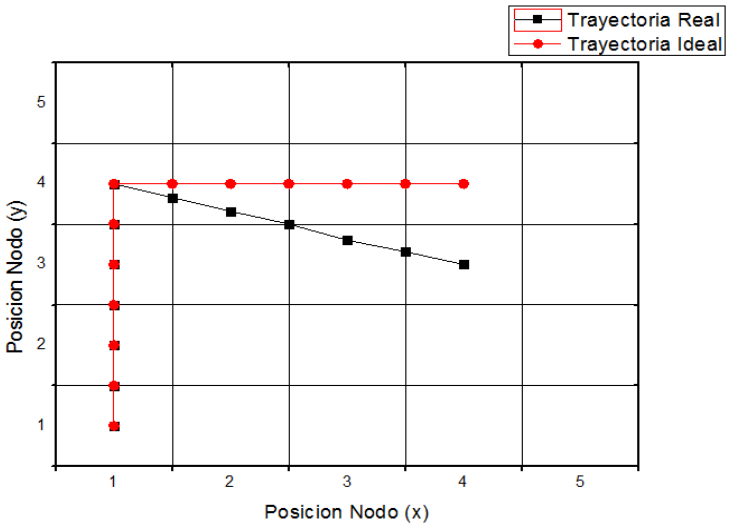
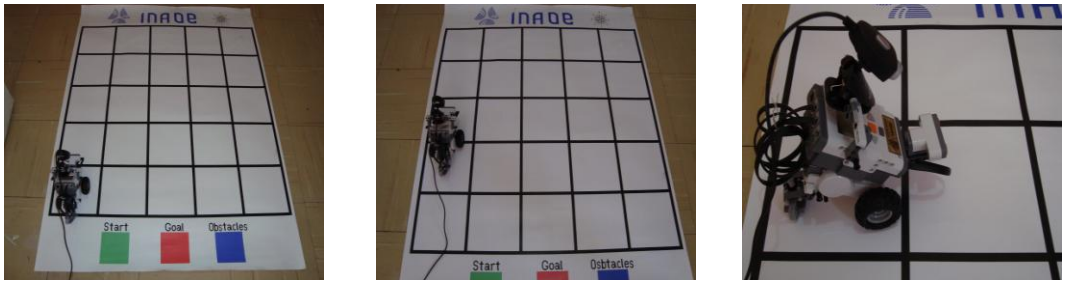


Fig. 4. 13 Comparación entre la trayectoria real y la ideal

4.4.2 Recorrido con ajuste de verticalidad por visión y control difuso

La segunda prueba consiste en repetir el experimento anterior agregando la rutina de corrección de verticalidad por visión y control difuso. La Figura 4.14 ilustra la ejecución del recorrido con la rutina de corrección agregada. La Figura 4.15 ilustra la comparación entre la trayectoria real y la entregada por el algoritmo de Dijkstra.



(a) Posición inicial del robot

(b) Inicio del recorrido de la trayectoria

(c) El robot se alinea en el primer giro de 90°

(d) Por cada nodo el robot revisa su alineación

(e) Segunda alineación en el recorrido

(f) Robot alineado en su posición final

Fig. 4. 14 Comportamiento del robot en el recorrido de la trayectoria con la rutina de corrección

Comparando los resultados obtenidos en las gráficas 4.13 y 4.15, se puede concluir que la rutina de corrección por visión y control difuso mejora considerablemente el recorrido del robot al lograr alinearlo cada vez que éste avanza de un nodo hacia otro. La Tabla 4.9 resume los resultados obtenidos en las pruebas de estos dos últimos experimentos.

Tabla 4. 9 Resultados del comportamiento del robot durante el recorrido de la trayectoria

Experimento	Total de Pruebas	Pruebas Exitosas	Pruebas Falladas	Porcentaje de éxito
Recorrido SIN la corrección por visión y control difuso	15	3	12	20%
Recorrido de la ruta CON la corrección por visión y control difuso	30	26	6	87%

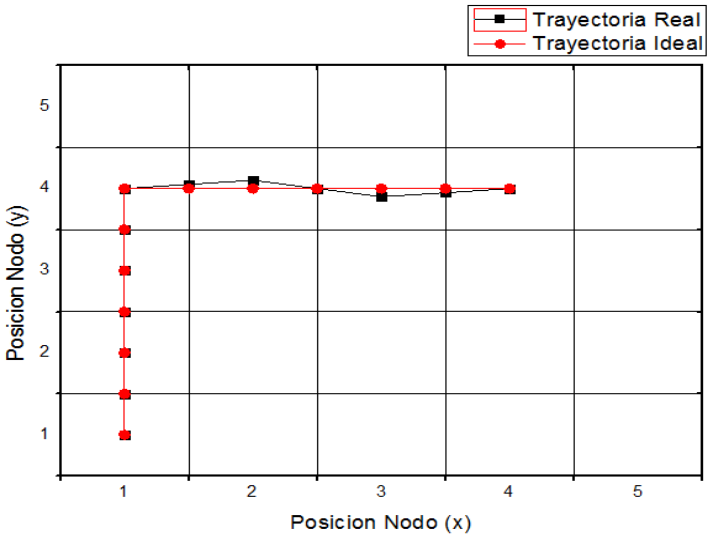
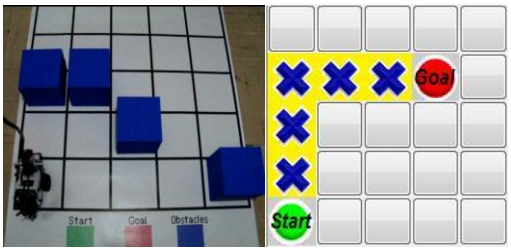


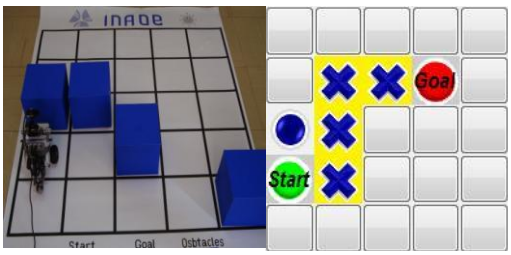
Fig. 4. 15 Comparación entre la trayectoria real y la ideal con la ejecución de la rutina de corrección

4.4.3 Recorrido automático por visión y ultrasonido

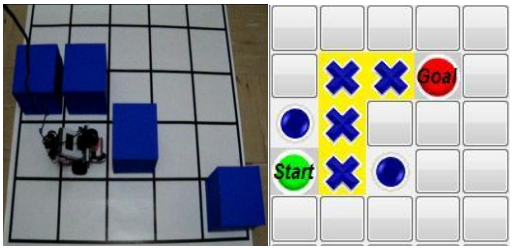
La tercera prueba consiste en evaluar el desempeño de la rutina de búsqueda automática de obstáculos. En esta prueba los obstáculos se colocan únicamente sobre el ambiente real y el algoritmo de Dijkstra entrega una trayectoria como si no hubiese ningún obstáculo presente en el mismo. Conforme el robot detecta los obstáculos, estos son plasmados sobre el ambiente virtual y la trayectoria es modificada. La alineación del recorrido es incluida en esta etapa. Las Figuras 4.16 y 4.17 muestran los resultados obtenidos.



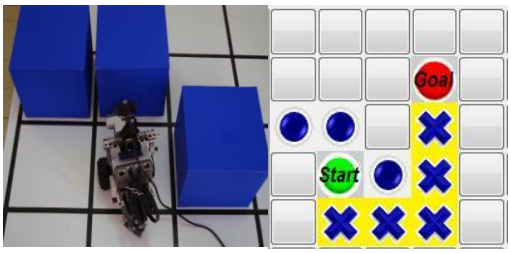
(a) Posición Inicial del robot y ruta entregada por el algoritmo de Dijkstra



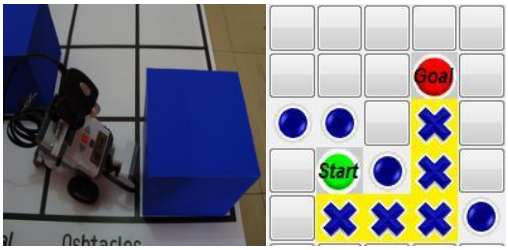
(b) Primer obstáculo detectado, la ruta se modifica



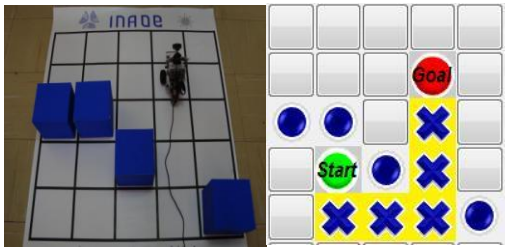
(c) Segundo obstáculo detectado, la ruta no se modifica ya que este no interfiere en la ruta.



(d) Tercer obstáculo detectado, la ruta se modifica nuevamente.



(e) Último obstáculo detectado, no interfiere en la ruta



(f) Posición final del robot, trayectoria final

Fig. 4. 16 Comportamiento en el recorrido automático por visión y ultrasonido

En el desarrollo de esta prueba, el robot logró detectar y reportar correctamente la posición de los obstáculos dentro del entorno. Así como también verificó y corrigió su recorrido para avanzar de un nodo hacia otro.

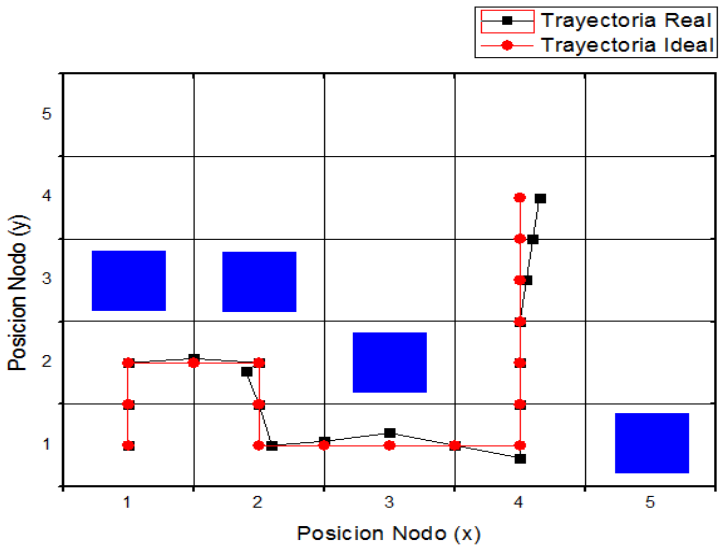


Fig. 4. 17 Comparación entre la trayectoria real y la ideal en el recorrido automático por visión y ultrasonido

4.5 Resumen de resultados

Con base a los resultados obtenidos en la evaluación de las diferentes etapas del sistema, se observa que en todas las pruebas el robot ha tenido la capacidad de alcanzar la meta establecida. La Tabla 4.10 muestra un resumen de los resultados obtenidos.

Tabla 4. 10 Resumen de resultados del sistema de navegación

Etapa	Total de Pruebas	Pruebas Exitosas	Pruebas Falladas	% Éxito
Implementación del Algoritmo de Dijkstra.	50	50	0	100%
Captura del Ambiente real	50	50	0	100%
Recorrido de la ruta más corta con la alineación correspondiente	30	26	4	87%
Recorrido automático por visión y Ultrasonido	20	17	3	85%

En todos los experimentos llevados a cabo, el algoritmo de Dijkstra estableció adecuadamente la ruta más corta, misma que fue completada con éxito en un 83% de las pruebas realizadas por el vehículo.

El sistema de control difuso controlado por visión le permitió al robot realizar los ajustes correspondientes, a efecto de mantener alineado al robot a lo largo de su recorrido. Cabe mencionar, que dadas las dimensiones del entorno real, aquellas rutas que el robot recorrió en más de 4 minutos se consideraron fracasadas.

Se realizó la traducción del ambiente real captado por la cámara hacia un ambiente virtual representado en forma de grafo. El algoritmo es insensible a la luz y a las sombras por lo que se obtienen resultados satisfactorios sin importar las condiciones de iluminación.

Por último, el robot logró explorar el entorno y reportó la localización de los obstáculos detectados por el sensor de ultrasonido. De igual manera, el tiempo de duración del recorrido es el criterio para determinar pruebas falladas.

4.6 Interfaz Gráfica de Usuario

Finalmente, el sistema se incorpora en una sola interfaz de usuario realizada bajo el ambiente GUI en Matlab. Con la finalidad de que el sistema sea fácil de manejar es necesario incluir las funciones principales dentro del diseño de la interfaz. De acuerdo con la distribución presentada en la Figura 4.18, se enlista una breve descripción de las funciones dentro de la interfaz:

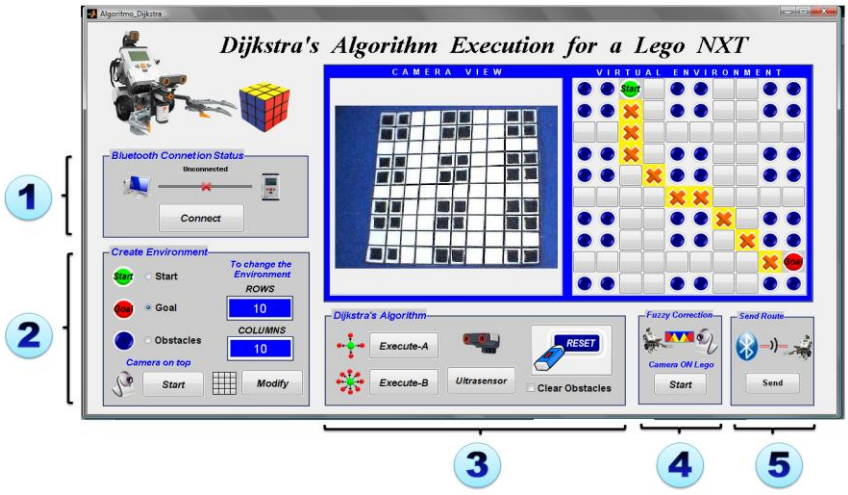
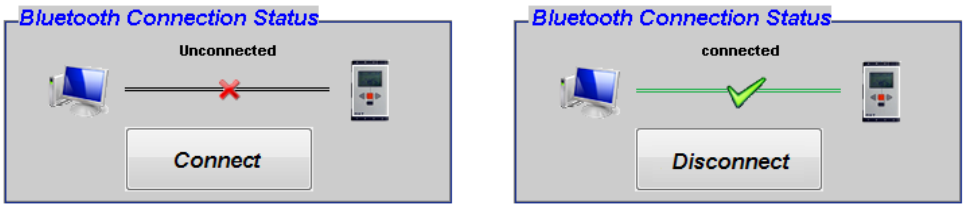


Fig. 4. 18 Interfaz gráfica de usuario

1. **Bluetooth Connection Status**. Habilita y deshabilita la comunicación entre el Lego y la computadora (ambas direcciones) usando la conexión inalámbrica Bluetooth.



(a) Conexión Deshabilitada (b) Conexión habilitada

Fig. 4. 19 Estados de la conexión Bluetooth

2. **Create Enviroment**. Permite establecer la posición del nodo INICIO *(Start)* y el nodo META *(Goal)*; así como también el número y posición de los obstáculos *(Obstacles)* en forma manual. Esta sección incluye una función para definir el número de filas y columnas existentes en el ambiente virtual. Este se modifica al pulsar la tecla *Modify*. En caso de no establecer el número y posición de los obstáculos, en la sección *Camera on top* se incluye la función de Captura del Ambiente real descrita en la sección 3.2.1.

Fig. 4. 20 Opciones para crear el ambiente virtual

3. **Dijkstra's Algorithm**. En esta sección se ejecuta el algoritmo de Dijkstra. *Execute-A* lo realiza con movimientos ortogonales y *Execute-B* con movimientos omnidireccionales. Cabe mencionar que estas dos funciones pueden ser ejecutadas con la conexión Bluetooth deshabilitada. En esta sección se incluyen

las siguientes operaciones: *Ultrasensor* la cual habilita el sensor ultrasónico montado en el robot y *Reset* borra la ruta trazada por el algoritmo y la ubicación de los nodos INICIO y META, permitiendo al usuario definir una nueva.

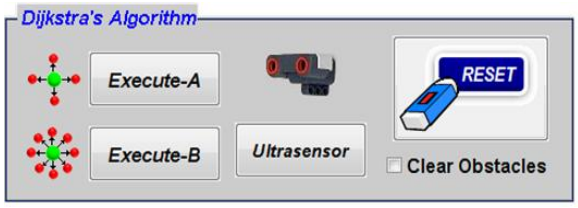


Fig. 4. 21 Opciones para ejecutar el algoritmo de Dijkstra

4. **Fuzzy correction**. Activa la cámara montada en el robot Lego NXT, ya que ésta es indispensable para realizar la corrección de la ruta.

5. **Send Route**. Envía la ruta establecida por el algoritmo hacia el robot Lego ejecutando la corrección por lógica difusa. Para llevar a cabo esta función es forzosamente necesario habilitar la conexión Bluetooth y la cámara montada en el robot.

En este capítulo se describieron las diferentes pruebas realizadas para validar el desempeño de las etapas del sistema de navegación propuesto. Se realizó la caracterización del sensor ultrasónico ante objetos colocados a pequeñas distancias. Así mismo, se comprobó el buen desempeño del algoritmo de captura automática de los obstáculos. Posteriormente se describieron distintas pruebas de navegación, las cuales permitieron evaluar la autonomía del robot ante la presencia de los obstáculos dentro de su entorno.

Capítulo 5

Conclusiones

5.1 Conclusiones generales

Se presentó un sistema de navegación autónoma basado en visión por computadora y control difuso, con orientación a aplicaciones de localización y mapeo simultáneo, implementado en su totalidad en Matlab, con la ayuda de las cajas de herramientas de lógica difusa y procesamiento de imágenes.

Se cumplió con el objetivo de encontrar la ruta más cercana y libre de colisiones en base al algoritmo de Dijkstra dentro de un mapa de localización tipo rejilla. Se implementó este algoritmo debido a su rápida ejecución y a que no requiere muchos recursos computacionales, tales como operaciones matemáticas complejas. El algoritmo implementado es capaz de encontrar y generar la ruta más corta desde un inicio hasta una meta con movimientos ortogonales y omnidireccionales.

Se logró detectar los obstáculos presentes en el entorno mediante el uso independiente de un sensor de ultrasonido y de una cámara para la captura automática de la información visual del entorno real en que se desplazará el robot.

De acuerdo con la caracterización del sensor de ultrasonido, este es capaz de detectar obstáculos a distancias entre 6 y 255 cm con un rango de apertura de 55° a 125°. Durante la ejecución de esta etapa, el robot fue capaz de detectar y retroalimentar la posición de los obstáculos dentro de su entorno.

El uso de técnicas de procesamiento de imágenes favoreció al buen desempeño de la detección automática de obstáculos; ya que estas facilitan la obtención de diferentes modelos de color. Como resultado, se obtuvo un algoritmo muy tolerante a los cambios de iluminación. Así como también, la segmentación de la imagen capturada permitió encontrar los obstáculos de manera satisfactoria y sistemática.

La corrección de la verticalidad en el robot se obtuvo mediante la implementación de un controlador difuso de una entrada y una salida. Como dato de entrada se requirió la información numérica de la posición de la línea de acotamiento detectada a través de la cámara web montada sobre el robot para entregar como dato de salida un nuevo dato numérico correspondiente al desplazamiento en grados en la trayectoria del robot.

Las funciones desarrolladas en este sistema pretenden servir como base en cursos de robótica y control para permitir a los estudiantes transferir los conocimientos teóricos a experimentos prácticos, así como contar con una plataforma para la incorporación de diversas tareas en aplicaciones de navegación autónoma de robots, tales como la identificación de salidas en un espacio cerrado, o el estacionamiento automático de un vehículo.

El sistema desarrollado permite también la experimentación con técnicas de localización y mapeo simultáneo, que podrían ejecutarse completamente en la computadora transfiriendo vía Bluetooth las señales de control para el desplazamiento del robot.

5.1.1 Trabajo Futuro

Este trabajo puede aumentar su capacidad funcional ya que es un sistema abierto a posibles mejoras, entre las cuales se recomiendan las siguientes:

- La incorporación de un algoritmo para el reconocimiento de patrones, el cual permita además de detectar los obstáculos poder reconocer la estructura geométrica de los mismos.
- Ampliar la capacidad de navegación y autonomía en ambientes no controlados, mediante la implementación de sistemas SLAM.
- Incorporar el uso de una cámara inalámbrica sobre el robot para mejorar su movilidad.

Apéndice A

Configuración de la conexión serial Bluetooth

La comunicación entre MATLAB y el LEGO Mindstorm NXT es a través de la interfaz serial Bluetooth. Como requisito inicial, la computadora debe tener un adaptador Bluetooth que sea compatible con el del Lego NXT. El sitio web http://.mindstorms.lego.com/Overview/Bluetooth.aspx proporciona la siguiente tabla de compatibilidad.

Tabla A.1 Compatibilidad con el Mindstorm NXT Bluetooth

Dispositivo Bluetooth	**Compatibilidad**
Abe UB22S	★★★
Belkin F8T003 ver. 2 (short range)	★★★
BlueFRITZ! AVM BT adapter, BlueFRITZ! USB v2.0	★★★
Cables Unlimited USB-1520	★★★
Dell TrueMobile Bluetooth Module	★★★
Dell Wireless 350 Bluetooth Internal Card	★
Dlink DBT-120	★★★
MSI Btoes	★★★
MSI StartKey 3X-faster	★★★
TDK GoBlue	★★★
Qtrek, Bluetooth USB Adapter v2.0	★★★

★★★ Compatible/Buen Funcionamiento
★ No Compatible / Pobre Funcionamiento

Para configurar la conexión entre MATLAB y el LEGO Mindstorm NXT es necesario seguir una serie de pasos:

1. Encienda el Lego NXT presionando el botón naranja.

Fig. A.1 Encender el NXT

2. En su computadora, inicie el software Bluetooth. Este típicamente tendrá una opción como "Buscando dispositivos Bluetooth" o "Nueva Conexión". Seleccione esta opción. Usted verá una lista de los dispositivos Bluetooth alrededor; seleccione el dispositivo nombrado "NXT" y "Siguiente".

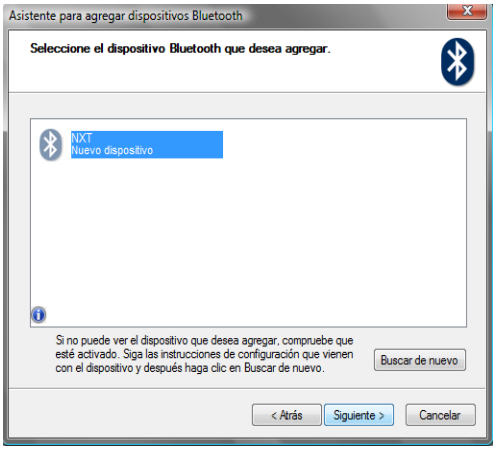


Fig. A.2 Selección del dispositivo NXT

3. En la siguiente ventana aparecerá la pregunta "¿Necesita una clave de paso para agregar su dispositivo?". Seleccione la opción "Usar la clave de paso que está en

la documentación". La computadora le pedirá una clave de paso o PIN, introduzca la clave que usted desee, por ejemplo: 1234.

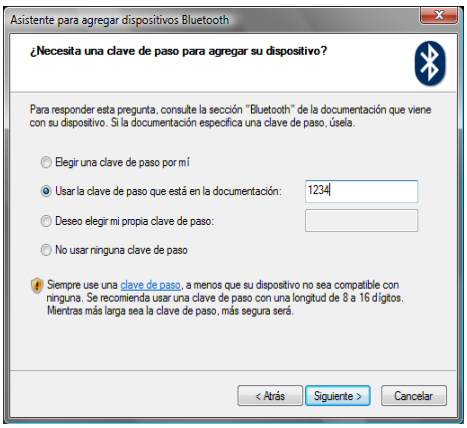
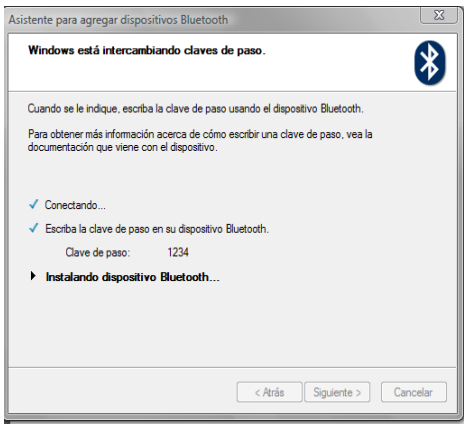


Fig. A.3 Selección de la clave de paso

4. La siguiente ventana desplegará el mensaje "Intercambiando claves de paso". En ese instante, el LEGO NXT emitirá un sonido y mostrará un mensaje:

Passkeys:
1234

Apriete el botón naranja para confirmar la clave de paso. Esto completará el "apareo" del Lego NXT con la computadora. Típicamente se necesita hacer esto solo una vez.



(b) Instalación del dispositivo Bluetooth

(a) Clave de paso en el Lego NXT

Fig. A.4 Intercambio de la clave de paso

5. El software Bluetooth desplegará la información acerca del puerto serial al cual el NXT ha sido asignado, por ejemplo: "COM 4" y "COM5". Es importante que conserve el nombre del puerto COM de salida, ya que será a través de este puerto por donde se envíen las órdenes al NXT desde Matlab.

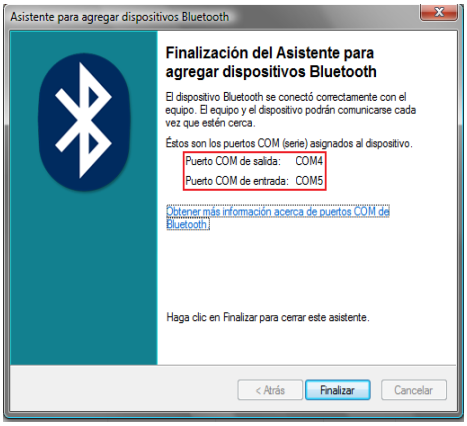


Fig. A.5 El dispositivo Bluetooth está agregado

6. Cada vez que se desee controlar al Lego NXT desde la computadora, asegúrese que su dispositivo Bluetooth este encendido. Una vez hecho, en el NXT seleccione el menú "Bluetooth" y oprima el botón naranja.

7. Seleccione el submenú "My Contacts" y usted podrá ver los contactos a los cuales el Lego se ha conectado con anterioridad. Seleccione el nombre de identificación de su computadora y apriete el botón naranja.

Fig. A.6 Menú "Bluetooth"

Fig. A.7 Submenú "My Contacts"

8. Sobre el display de la pantalla aparecerá la palabra "Connect". Apriete nuevamente el botón naranja para habilitar la conexión. En la parte superior izquierda de la pantalla usted puede observar el icono de Bluetooth. Este sirve para conocer el estado actual de la conexión:

El dispositivo Bluetooth está encendido, pero el NXT no es visible a otros dispositivos Bluetooth.

El dispositivo Bluetooth está encendido y el NXT es visible a otros dispositivos Bluetooth.

El dispositivo Bluetooth está encendido y el NXT está conectado a otro dispositivo Bluetooth.

Una vez que el símbolo  aparece, la conexión Bluetooth entre su computadora y el Lego NXT está habilitada.

Índice de Figuras

Índice de Tablas

Referencias

[1] O. Booij, B. Terwijn, Z. Zivkovic, B. Krose, "Navigation using an appearance based topological map", *IEEE International Conference on Robotics and Automation,* pp. 3927-3932, Rome, Italy, 10-14 April 2007.

[2] M. Usov, "Vision Based Mobile Robot Navigation", M.Sc. Thesis, *University of Twente EEMCS / Electrical Engineering Control Engineering*, June 2006.

[3] R. López Padilla, V. Ayala Ramirez , R. Sánchez- Yañez , "Some Experiments on Reactive Obstacle Avoidance for a Mobile Robot ", Universidad de Guanajuato, *18th International Conference on Electronics Communications and Computers, IEEE Computer Society,* pp. 193-196, Puebla, Mexico, 2008.

[4] Juan Manuel Ramírez, Pilar Gómez-Gil, Filiberto López Larios, "A Robot-vision System for Autonomous Vehicle Navigation with Fuzzy-logic Control using Lab-View", *Cuarto Congreso de Electrónica, Robótica y Mecánica Automotriz*, CERMA 2007, pp. 295-300, Cuernavaca, Morelos, 2007.

[5] I.J. García Enríquez, M. N. Ibarra Bonilla, J. M. Ramírez Cortés, P. Gómez Gil, "Seguimiento Autónomo de la Posición de un Objeto por Visión y Control Neuro-difuso en MATLAB", *6to. Congreso Internacional de Investigación en Ingeniería Eléctrica y Electrónica* (CIIIEE), Aguascalientes, México, Nov. 3-7, 2008.

[6] S. Thrun, D. Ferguson and et al., "A system for volumetric robotic mapping of abandoned mines", *Proceedings of the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[7] J.N. Wilson, "Guidance of agriculture vehicle: a historical perspective", Computers and Electronics in Agriculture, vol. 25, No. 2, pp. 3-9, 2003.

[8] Flan N.N., Moore K.L., "A small mobile robot for security and inspection", *Control Engineering Practice*, vol. 10, issue 11, 2004, pp. 1265- 1270. ISSN: 0967-0661.

[9] A. Ollero, "Robótica: Manipuladores y robots móviles", Alfaomega Marcombo, 2008.

[10] A. Benitez, C.J. Moreno, D. Vallejo, "Localization Control LEGO Robot's Navigation", *18th International Conference on Electronics Communications and Computers, IEEE Computer Society*, pp. 187-192, Puebla, Mexico, 2008.

[11] J. Fawcett, P. Robinson, "Adaptive routing for road traffic", *IEEE Computer Graphics and Applications*, vol. 20, No. 3, pp. 46-53, May-June 2000.

[12] H. Durrant-Whyte, T. Bailey, "Simultaneous localization and mapping (SLAM): Part I the essential algorithms", *Robotics and Automation Magazine*, vol 3, No. 2, pp. 99-110, 2006.

[13] M. Montemerl, S. Thrun, *"FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics"*. Germany, Springer, 2007.

[14] H. Durrant-Whyte, P. M. Newman, S. Clark, "A Solution to the Simultaneous Localization and Map Building (SLAM) Problem", *IEEE Transactions on Robotics and Automation*, vol. 17, No. 3, June 2001.

[15] P.M. , "The solution to the Simultaneous Localization and Map Building Problem", Ph. D. thesis, Dept. Mechanical Engineering, Australian Centre for Field Robotics, University of Sydney, Australia, March 1999.

[16] V. M. Jáquez, "Construcción de Mapas y Localización Simultánea con Robots Móviles", Tesis de Maestría, ITESM, Cuernavaca, 2005.

[17] S. Holmes, G. Klein, D.W. Murray, "A Square Root Unscented Kalman Filter for visual monoSLAM", *IEEE International Conference on Robotics and Automation, ICRA 2008*, pp. 3710-3716, Pasadena, Cal., USA, May 2008.

[18] A. Koning, "Shortest path algorithms based on Component Hierarchies", Ph. D. Thesis, University of Utrecht, department of Mathematics in cooperation with ORTEC, 2007.

[19] T.H. Comen and et al., *Introduction to algorithms*. Second Edition, Mc Graw Hill, 2002.

[20] M. Peñarada and et al., "Funcionamiento, representación y comportamiento de diferentes algoritmos frente al cálculo de un small world en ciencia", *9th International Society for Knowledge Organization Congress* (ISKO), Valencia, España, Marzo 2009.

[21] I. Hernández, C. Ochoa, "Control Difuso y construcción de un mecanismo capaz de golpear con distintos efectos una bola de billar". Tesis de Maestría, UDLAP, Cholula, Puebla, México, 2004.

[22] Y. Nikolaevna, "Automatic Estimation of Parameters to Reduce Rule Base of Fuzzy Control Complex Systems", Tesis de Maestría, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Agosto 2006.

[23] P. Dadone, "Design Optimization of Fuzzy Logic Systems", Ph.D. Thesis, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia, May 2001.

[24] J. Jantzen, *Foundations of Fuzzy Control*, Wiley, 2007.

[25] A. Ferreyra, "Redes Neuronales Difusas para modelado vía agrupamiento en línea: Aplicación a un condensador de aspiración". Tesis Doctoral, IPN, México, D.F, 2005.

[26] K. Passino and S. Yurkovich, "*Fuzzy Control"*, Addison-Wesley, 1998.

[27] E. H. Mamdani, S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller", *International Journal of Human-Computer Studies*, vol. 51, No. 2, pp. 135-147, 1975.

[28] T. Takagi, M. Sugeno, "Fuzzy Identification of systems and its application to modeling and control", *IEEE Transactions on Systems Man and Cybernetics*, vol. 15, No. 1, pp. 116-132, 1985.

[29] M. Thierry, V. Laurent, "Control laws for Takagi–Sugeno fuzzy models", *Fuzzy Sets and Systems*, vol. 120, No. 1, pp. 95-108, 2001.

[30] A. Moez, S. Mansour, C. Mohamed, M. Driss, "Takagi-Sugeno Fuzzy Control of Induction Motor", *International Journal of Electrical and Electronics Engineering*, vol. 2, No. 1, pp. 25-31, 2009.

[31] E. H. Mamdani and C.P. Pappis, "A Fuzzy Logic Controller for a Traffic Junction", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, No.10, pp. 707-717, 1977.

[32] E. Porras, "Diseño de un controlador difuso para un horno rotatorio de cemento". Tesis de Maestría. Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE), Septiembre 2005.

[33] C. Cardona, "Evaluación de algoritmos basados en lógica difusa aplicados al preproceso y detección de bordes en imágenes digitales". Tesis de Licenciatura. Universidad Nacional de Colombia, Colombia, 2004.

[34] A. Saffiotti, "The Uses of Fuzzy Logic in Autonomous Robot Navigation". *Soft Computing: A Fusion of Foundations, Methodologies and Applications*, vol. 1, No. 4, pp. 180-197, 1997.

[35] A. Saffiotti, D. Driankov, "Fuzzy Logic Techniques for Autonomous Vehicle Navigation". Physica-Verlag: A springer Company, 2001.

[36] M. Sugeno and M. Nishida, "Fuzzy control of model car", *Fuzzy Sets and Systems*, vol. 16, No. 2, pp. 103-113, 1985.

[37] T. Takeuchi, Y. Nagai, N. Enomoto, "Fuzzy control of a mobile robot for obstacle avoidance", *Information Sciences: an International Journal*, vol. 45, No. 2, pp. 231-248, 1988.

[38] A. Saffiotti, K. Konolige, E. H. Ruspini, **"**A multivalued-logic approach to integrating planning and control", *Elsevier Science: Artificial Intelligence*, vol. 76, pp. 481-526, 1995.

[39] C. Kim, M. Mohan, "A Neuro fuzzy Controller for Mobile Robot Navigation and Multirobot Convoying", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28, No. 6, pp. 829-840, December 1998.

[40] R. Young-Jae, L. Young-Cheol, "Neuro Fuzzy Control System for Vision.based autonomous vehicle", *Fuzzy System Conference IEEE International*, vol. 3, 1999, pp. 1643-1648.

[41] D. Forsyth, J. Fonce, *Computer vision: a modern approach*. Upper Saddle River: Prentice Hall, 2003.

[42] D. Song, H. N. Lee, J. Yi, A. Levandowski, "Vision-based motion planning for an autonomous motorcycle on ill-structured roads", *Auton Robot*, vol. 23, No. 3, pp. 197-212, October 2007.

[43] Lego Mindstorm NXT Software for Matlab and Simulik. http://www.mathworks.com/programs/mindstorms/