UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

TESIS

“CONTROL DE UN ROBOT MÓVIL APLICANDO CONTROL DIFUSO Y VISIÓN ARTIFICIAL DESARROLLADO EN LABVIEW.”

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA:

HUGO ISRAEL ALCARAZ HERRERA

DIRECTORA:
MTRA. GLORIA CORREA PALACIOS

CIUDAD UNIVERSITARIA Marzo 2012

Dedicatoria

A los que no están pero siempre me acompañan:

> ***A mi madre**, una mujer sencilla, noble, abnegada, carismática y con un enorme corazón. Sin ella seguramente no estaría aquí ahora, principalmente a ella, en donde quiera que esté. Al tío Mario, un ser humano del cuál aprendí a estar conmigo mismo y a disfrutar de la compañía de la soledad. A Oscar, un gran amigo el cual siempre estuvo conmigo en las buenas y en las malas; siempre escuchó y nunca juzgó. Y a mi padre, un buen hombre.*

A mi adorada familia:

> *A Dianela, mi hermana, que con su apoyo, palabras, cariño, compañía y solidaridad, ha contribuido a ser la persona que ahora soy. A mis dos queridas y maravillosas tías Liz y Luz que con su constante e incondicional apoyo desde que yo era un niño, he podido realizar todas mis metas. A mis queridos tíos Miguel, Toño, Donato y Andrés, de quienes aprendí el valor del trabajo y que la constancia, el empeño y la visión son la clave de todo éxito. A mamá Priscila, a papá Fortunato, a mis fabulosos tíos Arturo, Fernando, Salvador y Alejandro que me cuidaron en la infancia y me enseñaron lo que es ser sencillo, justo y agradecido. A mis primos Miguel, Julio y Zaray, con quienes crecí y pasé (y seguiremos pasando) grandes momentos de júbilo y diversión que crearon un fuerte vínculo entre ellos y yo. A mis sobrinos Nori, Sebastián, Mildred, Amelia, Felipe y Emre, a quienes les brindaré mi apoyo, atención y dedicación en lo que necesiten, tal y como en su tiempo me fueron brindados.*

A mi otra familia, mis amigos:

> *A Enrique Martínez, a los gemelos Jorge y David Zamora, Tonatiuh López, Aldo Angulo, Javier Flores, José Luis Jerónimo, Carlos Limón, Ulises López, Sandra Navarrete, Giovanna Martínez, y los que no mencioné pero que son importantes también, que con sus consejos, compañía, comprensión, cariño, alegría y humor he podido forjar mi personalidad y carácter.*

Dedico también este trabajo a todas las personas que estuvieron, están y estarán presentes en mi vida y que me dieron, me dan y me darán algo que me enriqueció, me enriquece y me enriquecerá como persona. De igual forma agradezco a aquellas personas que alguna vez no me hicieron o desearon algún bien porque gracias a eso, me hice más fuerte y resistente.

¡Gracias a todos y a cada uno de ustedes!

Agradecimientos

A la Universidad Nacional Autónoma de México (UNAM) por cobijarme como un hijo y darme la mejor formación que pude recibir. ¡Agradezco infinitamente a mi Alma Mater!

A la Facultad de Ingeniería y a cada uno de los profesores con los que tomé clase, por proporcionarme los conocimientos necesarios para ser un ingeniero íntegro y competente.

Al Departamento de Ingeniería de Control y Robótica de la Facultad de Ingeniería por su generoso apoyo en el financiamiento de esta tesis.

A la Mtra. Gloria Correa Palacios, por sus consejos y comprensión brindados, por su siempre paciente y constante apoyo en el desarrollo de esta tesis y sobre todo, por su confianza y amistad.

A mis sinodales M. I. Ricardo Garibay Jimenez, M. I. Luis César Vázquez Segovia, M. I. Aurelio Adolfo Millán Nájera y al Ing. Rubén Anaya García por su profesionalismo al trabajar y el apoyo dado en la corrección y comentarios de esta tesis, por su gran calidad humana y su incansable afán de transmitir el conocimiento. ¡Gracias profesores por su tiempo!

Al Ingeniero Vicente Flores y al Ingeniero Moisés Rueda un agradecimiento especial porque fueron un importante y gran apoyo en momentos de estancamiento en el desarrollo de esta tesis. ¡Gracias por su apoyo!

“No hay hombre tan grande como sus propias ideas. En consecuencia, cualquier máquina que encarne una idea es más grande que el hombre que la hizo... ¿Y qué hay de malo en eso?”

Ray Bradbury.

Índice

Introducción.

Hoy día, los robots forman parte elemental en las labores cotidianas del humano; son usados en tareas repetitivas, sucias, difíciles y peligrosas. La robótica ha tenido un desarrollo tan rápido e intenso, que en pocos años ha alcanzado metas que en el pasado se creían temas de ciencia ficción. Los robots son muy utilizados en plantas de manufactura, montaje, transporte, exploraciones en la Tierra y en el espacio, cirugía, armamento, investigación de laboratorios, entre otros, siendo la industria, el principal mercado donde los robots son utilizados.

Debido a la gran variedad de aplicaciones que tiene los robots, surge la idea de la presente tesis, donde se desarrolla un prototipo de robot móvil, que fue construido con el apoyo del Departamento de Ingeniería de Control y Robótica de la Facultad de Ingeniería, capaz de recolectar un cierto número de objetos en un área específica por medio de visión artificial (reconocimiento de patrones), gracias a sus características de movilidad y sujeción. En la construcción del robot móvil se empleó el set de recursos Lego Mindstorms que es una plataforma para desarrollar y diseñar robots de una forma rápida y sencilla, con piezas diseñadas para ensamblarse sin necesidad de soldar o atornillar, el cual cuenta con un microcontrolador que tiene puertos de comunicación para controlar sensores y actuadores de forma sencilla. La programación del prototipo fue desarrollada en LabVIEW, que es una herramienta gráfica que permite diseñar, probar y desarrollar sistemas con el uso del lenguaje de programación G, es decir, programación gráfica, lo que hace que el desarrollo de software sea intuitivo, rápido y eficaz. El lazo de control del prototipo fue desarrollado utilizando los conceptos de lógica difusa y máquina de estados.

En el Capítulo Uno se explica el hardware del robot móvil consistente en la configuración de su sistema de locomoción, sujeción, actuadores y sensores. En el Capítulo Dos se describe la teoría aplicada para controlarlo. El Capítulo Tres menciona el entorno de programación, módulos y recursos de software de LabVIEW aplicados. El Capítulo Cuatro es la parte medular de esta tesis, ya que en ella se describe la arquitectura del sistema de control que rige el comportamiento del prototipo. Finalmente, el Capítulo Cinco muestra los resultados de las pruebas realizadas al sistema, así como una posible mejora del mismo.

El objetivo general de la tesis consiste en controlar un robot prototipo aplicando la teoría de control difuso y visión artificial en entorno de programación LabVIEW.

Capítulo 1
Hardware utilizado.

1.1 Robot móvil.

El término *robot*, fue acuñado en 1920 y procede de la palabra checa *robota*, cuyo significado es “ciervo” o en ruso, “trabajo obligatorio”. Un robot es una máquina programable que puede manipular objetos y realizar operaciones que antes sólo podían hacer los humanos, puede ser un sistema electromecánico o puede ser un sistema virtual. La Organización Internacional para la Estandarización (ISO) provee una definición de robot en la ISO 8373: “un manipulador automáticamente controlado, reprogramable, multiuso, programable en tres o más ejes que pueden estar fijos en un lugar o movilizarse para ser usado en aplicaciones de automatización industrial”.

La evolución de los procesos naturales y la tecnología llevó al origen de los robots móviles los cuáles son aquellos que tienen la capacidad de desplazarse a través del medio para el cual están diseñados. Los primeros robots móviles aparecen como solución al problema de transporte eficaz para los materiales entre las distintas zonas de la cadena de producción de las industrias. Los primeros en aparecer son los vehículos guiados automáticamente (Automatic Guided Vehicle) y que se mueven por rutas previamente establecidas. El desarrollo de gran parte de los robots móviles se encuentra limitado a entornos donde la navegación se realiza con una capacidad sensorial y de razonamiento mínimas. La tarea consiste en realizar una serie de acciones que al término, el vehículo supone que ha alcanzado su objetivo, ante cualquier cambio en el entorno de trabajo que afecte a la navegación, el vehículo será imposibilitado para ejecutar acciones opcionales que le permitan retomar su tarea.

La tendencia actual va dirigida hacia el robot móvil autónomo, el cuál tiene la capacidad de moverse en entornos no estructurados y que tiene conocimiento incierto. Todo reconocimiento del entorno se logra mediante la interpretación de datos suministrados a través de sensores y del estado actual del robot móvil. El robot móvil autónomo tiene una característica principal: una relación “inteligente entre las operaciones de percepción y de acción que definen su comportamiento y le permite llegar al logro de su tarea. La autonomía depende directamente de la capacidad del robot para obtener datos del entorno y convertirlos en órdenes de tal forma que aplicados a los actuadores, garantice la realización de su tarea.

Las principales características que diferencian a un robot móvil autónomo de cualquier otro son:

• *Percepción*.- Determina la relación del robot con el entorno mediante el uso de sensores.

• *Razonamiento*.- Determina las acciones que deben realizarse en cada momento según el estado del robot y su entorno para finalizar la tarea.

Los robots móviles se clasifican de acuerdo a la forma de desplazamiento o al medio en el cual trabajan. En base a esto, existen robots móviles de ruedas, patas, orugas, aéreos, marinos, espaciales, etc. Y también pueden clasificarse por su tamaño: robots, minirobots y microrobots.

El robot móvil utilizado para la realización de alguna tarea está definido por el tipo de entorno en el que se desenvolverá; para entornos en donde el piso es plano, se usan robots de ruedas. Si el robot necesita moverse por escaleras o pisos muy irregulares, un robot con patas sería lo ideal. Sin embargo, si el terreno es muy irregular un robot de orugas puede ser la elección. Cabe mencionar que para construir un robot móvil no es necesario hacerlo desde cero; puede usarse un sistema comercial y agregarle lo necesario para que pueda realizar la tarea sin problemas.

En resumen, un robot móvil autónomo es un dispositivo compuesto de sensores que reciben datos de entrada y que puede estar conectado a una estación de control, ésta puede recibir los datos de entrada y determinar la acción que debe realizar el robot.

Una de las clasificaciones más comunes en la robótica móvil se basa en el sistema de locomoción que usan los robots para desplazarse. A continuación se describirán las características más sobresalientes de los sistemas más comunes.

Vehículos con ruedas

Los vehículos con ruedas son la solución más simple y eficaz para conseguir la movilidad en terrenos muy duros y que no permiten deslizamientos. Algunas configuraciones de este tipo de vehículos son:

• Triciclo clásico

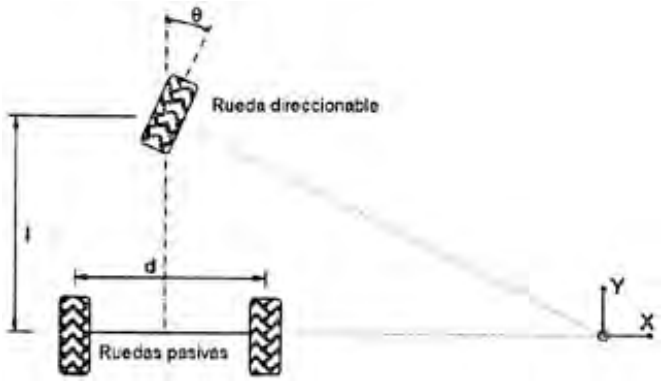

Figura 1.1 Configuración de triciclo clásico

En ese sistema, la rueda delantera sirve para la tracción y para el direccionamiento. El eje trasero, con dos ruedas laterales, es pasivo y sus ruedas se mueven libremente (Figura 1.1). La maniobrabilidad es muy buena pero puede presentar problemas de inestabilidad en terrenos irregulares y difíciles. El centro de gravedad tiende a desplazarse cuando el vehículo

se desplaza por una pendiente, lo que provoca una pérdida en la tracción. Debido a su simplicidad, es usado comúnmente en vehículos para interiores y exteriores pavimentados.

• Ackerman

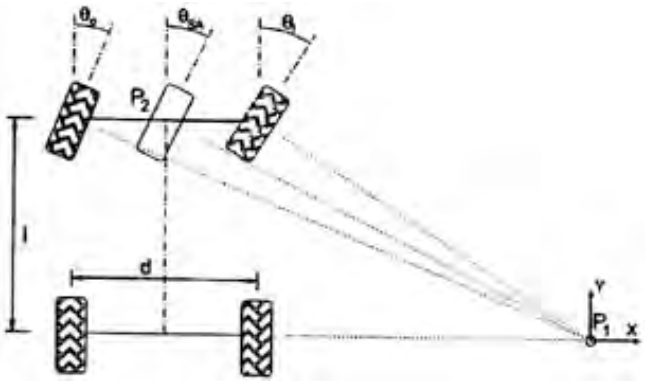

Figura 1.2 Configuración de Ackerman

Es el sistema normalmente en vehículos de cuatro ruedas. En este sistema, la rueda delantera interior gira un ángulo ligeramente superior al exterior para eliminar el deslizamiento. Las prolongaciones de los ejes de las dos ruedas delanteras, intersectan en un punto sobre la prolongación del eje de las ruedas traseras. El lugar de los puntos trazados sobre el suelo por los centros de los neumáticos son circunferencias concéntricas con centro en el eje de rotación. Si se desprecian la fuerzas centrífugas, los vectores de velocidad instantánea son tangentes a estas curvas. El mayor problema de este tipo de configuración es la limitación en la maniobrabilidad (Figura 1.2).

• Direccionamiento diferencial

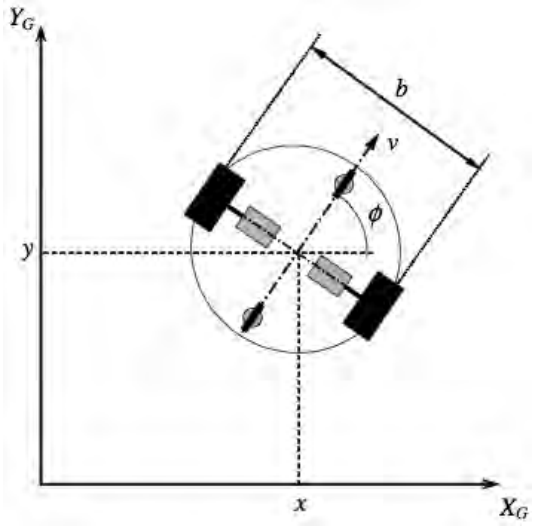

Figura 1.3 Configuración de Direccionamiento diferencial

El direccionamiento viene dado por la diferencia de velocidades de las ruedas laterales. Dos ruedas montadas en un único eje son independientemente propulsadas y controladas. La tracción se consigue también con estas mismas ruedas. Adicionalmente, existe una o más ruedas para soporte. Esta configuración es muy útil si consideramos la habilidad del móvil para cambiar su orientación sin necesidad de movimientos de translación. Se usa frecuentemente en robots que se desplazarán en interiores (Figura 1.3).

• Skid Steer

Figura 1.4 Configuración de Skid Steer

Se disponen de varias ruedas en cada lado del vehículo que actúan de forma simultánea. El movimiento es el resultado de combinar las velocidades de las ruedas de la izquierda con las de la derecha. Estos robots se han utilizado para la inspección y la obtención de mapas de tuberías encerradas empleando sistemas de radar, también se han utilizado en aplicaciones mineras y en misiones de exploración espacial no tripuladas (Figura 1.4).

• Pistas de deslizamiento

Figura 1.5 Configuración de Pistas de deslizamiento

Son vehículos tipo oruga en los que la impulsión y el direccionamiento se consiguen mediante pistas de deslizamiento. Pueden considerarse en cuestión de funcionamiento, parecidas a la configuración *Skid Steer*. Las pistas actúan de forma análoga a rudas de gran diámetro. La locomoción mediante pistas de deslizamiento es muy útil en navegación a campo abierto o terrenos irregulares. En esta configuración, la impulsión está menos limitada por el deslizamiento y la resistencia al desgaste es mayor (Figura 1.5).

• Ruedas síncronas

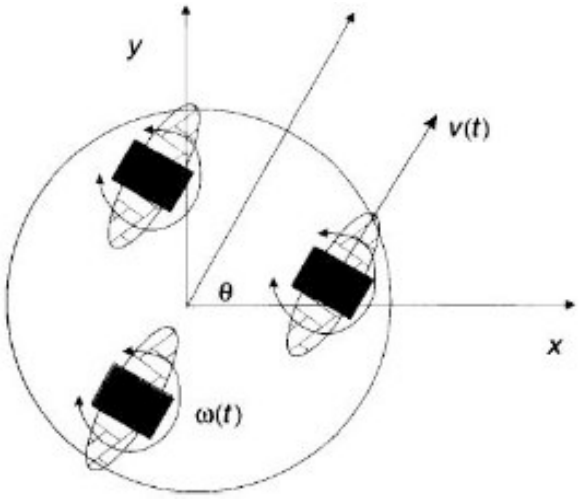

Figura 1.6 Configuración típica de Ruedas síncronas

Consiste en la actuación simultánea de todas las ruedas, que giran de forma síncrona. La transmisión se consigue mediante coronas de engranajes con correas concéntricas. En un sistemas de ruedas síncronas, cada rueda es capaz de ser conducida y dirigida (Figura 1.6). Algunas de las configuraciones más usadas son:

- Tres ruedas directrices se montan acopladas en los vértices de un triángulo equilátero, en la mayoría de las veces, debajo de una plataforma cilíndrica.
- Todas las ruedas propulsan y giran al unísono.

Este tipo de configuración puede controlar sin problema alguno la orientación del eje de rotación.

• Tracción omnidireccional

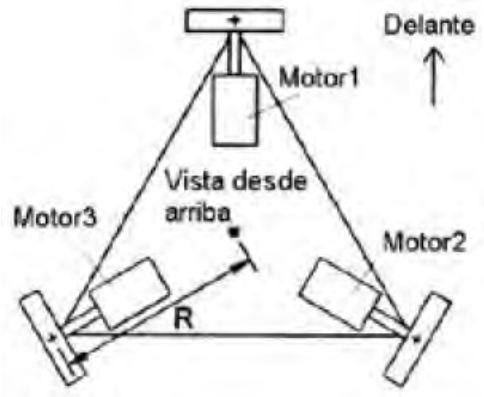

Figura 1.7 Configuración de Tracción omnidireccional

Esta configuración se basa en la utilización de tres ruedas directrices y motrices. Tiene tres grados de libertad, por lo que puede realizar cualquier movimiento y posicionarse en cualquier posición y en cualquier orientación. No presenta limitantes cinemáticas (Figura 1.7).

Vehículos con patas

Permiten aislar el cuerpo del robot del terreno empleando únicamente puntos discretos del soporte. Es posible adaptar el polígono de soporte para mantener la estabilidad y pasar sobre obstáculos. Este tipo de vehículos resulta ideal para terrenos muy irregulares. Así mismo, mediante las patas, es posible conseguir la omnidireccionalidad y el deslizamiento es mucho menor.

En los robots con patas, el mecanismo de locomoción es mucho más complejo y el consumo de energía aumenta considerablemente. Los problemas de planificación y control son más complicados que en los vehículos con ruedas (Figura 1.8).

La configuración más común es la de seis patas pero actualmente la tendencia va dirigida a los robots bípedos, que emulan el sistema de locomoción del hombre. En estas aplicaciones existen aplicaciones como los robots de patas utilizados para escalar o trepar, robots para exploración, etc.

Figura 1.8 Configuración típica de los vehículos con patas

Vehículos submarinos y aéreos

El interés de tales robots surge como necesidad para tareas como inspección, recolección de información o mantenimiento de instalaciones en entornos naturales en los que el acceso resulta muy riesgoso o complicado para el humano. Estos vehículos son el resultado de la evolución de vehículos que normalmente era controlados por un piloto (Figura 1.9).

Figura 1.9 Robot submarino “Super Mohawk II”

1.2 Lego Mindstorms NXT.

La línea Lego Mindstorms NXT, nació a partir de un acuerdo entre Lego y el MIT (Massachussetts Istitute of Technology). En este trato, Lego financiaría investigaciones del grupo de epistemología y aprendizaje del MIT sobre cómo aprenden los niños y a cambio obtendría nuevas ideas para sus productos que podría lanzar al mercado sin pagar regalías al MIT.

De este acuerdo, surge Lego Mindstorms, un sistema integrado de robótica con partes electromecánicas controladas por computadoras, desarrollado por Lego. El sistema posee elementos básicos de las teorías robóticas como la unión de piezas y la programación de acciones de forma interactiva.

El sistema básico de Lego Mindstorms (Figura 1.10) incluye:

• Bloque NXT o brick (microcontrolador)

• Sensor de tacto o toque

• Sensor de sonido

• Sensor de luz

• Sensor de color

• Sensor ultrasónico

• Servomotores

Figura 1.10 Kit básico Lego Mindstorms NXT

Bloque NXT

Figura 1.11 Bloque NXT

El bloque NXT dispone de 3 puertos para servomotores y 4 puertos para para sensores. Los puertos no son como los de un microcontrolador común, las entradas y salidas son conexiones para un tipo de cable estándar de Lego.

Dispone de una entrada USB de hasta 12 Mb/s para cargar y descargar programas, ficheros, etc. Tiene también comunicación Bluetooth (Bluetooth Clase II V. 2.0). En la parte posterior del bloque está la alimentación de energía que puede ser con 6 pilas AA o una batería recargable, además tiene un botón de reset. En la parte frontal tiene un LCD de 100 x 64, cuatro botones que también pueden utilizarse como entradas.

El componente central del bloque NXT es un microcontrolador ARM7 de 32 bits con 256 KB de memoria flash y 64 KB de memoria RAM, con una velocidad de 48 MHz. Tiene un segundo microcontrolador AVR de 8 bits con 4 KB de memoria flash y 512 B de memoria RAM, con una velocidad de 4 MHz (Figura 1.11).

Sensores

Los sensores permiten al robot NXT tener una visión del mundo exterior y cómo responder a él. Con este fin, el Kit NXT viene con cuatro tipos de sensores. A continuación, se describirán brevemente algunos de ellos:

• Sensor de tacto o toque

Figura 1.12 Sensor de tacto NXT

El sensor de tacto o toque permite al robot NXT tener un sentido del tacto. Este sensor le comunica al bloque NXT cuando ha sido presionado o liberado. Esta información puede resultar importante en la navegación o manipulación de objetos. Este sensor tiene varios puntos de montaje en función del tipo de contacto que se desea detectar. El bloque NXT sólo permite usar 2 sensores de tacto o toque en el robot a la vez (Figura 1.12).

• Sensor de sonido

Figura 1.13 Sensor de sonido NXT

El sensor de sonido permite al robot NXT tener un sentido de audición. El sensor es capaz de detectar en dos modos diferentes. El primer modo se le conoce como “decibeles ajustados” (en inglés *adjusted decibels*, dbA) que imita la forma en cómo el oído humano percibe el sonido ambiental, esto quiere decir que la sensibilidad del dispositivo se adapta a la sensibilidad del oído humano, es decir, ignora las frecuencias muy altas o muy bajas. El otro modo es el llamado “decibel estándar” (en inglés *standard decibel*, db) que simplemente registra la frecuencia del sonido por igual.

El sensor de sonido devuelve los resultados como un porcentaje, puede medir hasta 90 decibeles. La siguiente tabla es un ejemplo de cómo se asocia el porcentaje resultante con sonidos del medio ambiente.

Porcentaje (%)	Descripción
0 - 5	Habitación en silencio
5 - 10	Personas hablando lejos
10 - 30	Conversación normal
30 - 100	Personas gritando, música a alto volumen

Tabla 1.1 Tabla de porcentajes y sonidos del sensor de sonido NXT

Este sensor no está diseñado para utilizarse en aplicaciones de reconocimiento de voz o similares y no permite un control preciso del robot mediante el sonido (Figura 1.13).

• Sensor de luz

Figura 1.14 Sensor de luz NXT

El sensor de luz es uno de los sensores que da visión al robot NXT. Este sensor puede leer los niveles de luz ambiental o la luz que se refleja en una superficie. Al igual que el sensor de tacto, el sensor de luz puede jugar un papel determinante en la navegación del robot NXT. El sensor de luz también trae integrado una pequeña lámpara que emite luz roja que es de gran utilidad para la iluminación de ambientes oscuros o para amplificar la luz reflejada (Figura 1.14).

• Sensor de color

Figura 1.15 Sensor de color NXT

El sensor de color es una mejora al sensor anterior, es decir, además de ser sensor de color, duplica la funcionalidad del sensor de luz original. Este sensor detecta el color de forma secuencial, midiendo la cantidad de luz reflejada en los leds rojo, verde y azul. Las tres

mediciones se comparan para determinar el valor del color. La tabla siguiente muestra los valores y colores correspondientes.

Valor	Color
1	Negro
2	Azul
3	Verde
4	Amarillo
5	Rojo
6	Blanco

Figura 1.2 Tabla de valores y colores del sensor de color NXT

Cuando se activa el sensor de luz, sólo se utiliza uno de los leds (o ninguno de ellos) al realizar las mediciones del nivel de luz ambiental. El led tricolor se puede usar para depuración de color o como una lámpara ornamental (Figura 1.15).

• Sensor ultrasónico

Figura 1.16 Sensor ultrasónico NXT

Al igual que el sensor de luz, el sensor ultrasónico da visión al robot NXT. Este sensor funciona emitiendo ondas de sonido para que reboten en una superficie. Se mide la distancia calculando el tiempo que tarda la onda sonora en regresar después de rebotar en la superficie. Superficies grandes y planas son las más fáciles de detectar, objetos redondos o delgados son muy difíciles de detectar. El sensor ultrasónico tiene un rango de operación de 0 a 255 cms con un error de ±3 cms. El bloque NXT sólo permite usar 1 sensor ultrasónico en el robot a la vez (Figura 1.16).

Es importante mencionar que existen algunos otros sensores que no son de uso tan común, algunos de ellos son:

• Sensor de temperatura

• Brújula

• Acelerómetro

Estos dos últimos, son fabricados y distribuidos por *HiTechnic*.

Actuadores

Una de las descripciones de un actuador y quizá la más aplicable a este proyecto es la de un dispositivo generalmente mecánico cuya función es proporcionar fuerza para mover otro dispositivo mecánico. La fuerza que provoca al actuador puede venir de: presión neumática, presión hidráulica o fuerza motriz eléctrica. Así, Lego Mindstorms NXT cuenta con un sólo tipo de actuador:

• Servomotor

Figura 1.17 Servomotor NXT

Los motores NXT son servos, es decir, su posición interna y el estado en el que se encuentra puede ser controlado por alguna unidad externa. Esto es posible usando el sensor de rotación incorporado, permitiendo al bloque NXT controlar el servomotor con precisión. Se puede controlar el giro con una precisión de hasta un grado o correr a una velocidad determinada. Es posible sincronizar dos motores fácilmente, lo que permite un desplazamiento recto (Figura 1.17).

Capítulo 2
Teoría de Control aplicada al Robot.

2.1 Lógica Difusa.

Introducción.

También conocida como *fuzzy logic,* la lógica difusa es una metodología que permite manejar información imprecisa, vaga o ambigua: “Mucha fuerza”, “muy rápido”, “un poco frío” y a partir de ésta, obtener una conclusión de forma simple y elocuente. La lógica difusa emula la forma en cómo los humanos razonan, es decir, emula la forma en cómo una persona toma una decisión, basándose en la información con características mencionadas anteriormente.

Lotfi Asker Zadeh, un eminente profesor de la Universidad de California, Berkeley, fue quien en 1965 desarrolló el concepto de Lógica Difusa basado en los *conjuntos difusos*. Dichos conjuntos permiten que un elemento siempre tenga un cierto grado de pertenencia a un conjunto, en comparación con los conjuntos clásicos, que sólo permiten al elemento formar parte o no de un conjunto.

Conjuntos clásicos.

Un conjunto es una colección de objetos que tienen algo en común, formalmente un conjunto clásico A en un universo de discurso U se puede definir de diversas formas: Enumerando los elementos que pertenecen al conjunto, especificando las características que deben cumplir los elementos que pertenecen a ese conjunto o en términos de la llamada función de membresía o pertenencia $\mu_A(x)$:

$$\mu_A(x) = \begin{Bmatrix} 1 \text{ si } x \in A \\ 0 \text{ si } x \notin A \end{Bmatrix}$$

Los elementos del universo del discurso *U* pueden o no pertenecer al conjunto *A*, mostrado en la Figura 2.1:

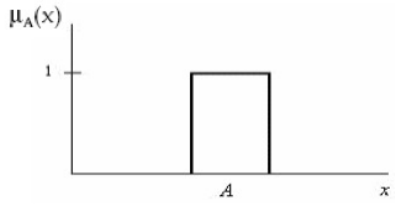

Figura 2.1 Función característica de un conjunto clásico

Conjuntos difusos.

Un conjunto difuso *A* en el universo del discurso *U*, tiene como característica que su función de membresía o pertenencia puede tomar valores en el intervalo [0,1] y puede representarse como un conjunto de pares ordenados de un elemento *x* y su valor de pertenencia al conjunto:

$$A = \{(x, \mu_A(x)) | x \in U\}$$

o en otros términos:

$$A = \int\{\mu_A(x)|x\} \text{ ó } \sum\{\mu_A(x)|x\} \text{ para } x \in U$$

Éstas dos últimas para los casos continuos y discretos respectivamente. Dentro de la lógica difusa, los símbolos $\int$ y $\sum$ hacen referencia o implican a un conjunto difuso y no a las operaciones de integración y sumatoria como habitualmente se conocen.

La función de pertenencia es una simple medida del grado con que *x* pertenece al conjunto A. Esto se representa de la siguiente manera:

$$\mu_A(x): U \rightarrow [0,1]$$

Muchos de los conceptos que la teoría clásica de conjuntos se pueden llevar a la teoría de conjuntos difusos; algunos de los más recurrentes son:

• *El soporte de un conjunto difuso A* en el universo del discurso U es un conjunto que contiene todos los elementos de U que tienen un valor de pertenencia distinto de 0, es decir:

$$sop(x) = \{x \in U | \mu_A(x) > 0\}$$

Si el soporte de un conjunto no contiene elemento, entonces hay un conjunto difuso vacío. Si el soporte de un conjunto difuso es un punto, entonces es un “singleton” difuso (Figura 2.2).

Figura 2.2 Soporte de un conjunto difuso

• *El punto de cruce de un conjunto difuso* es el punto de U cuyo valor de pertenencia del conjunto es igual a 0.5.

• Dos conjuntos A y B son iguales si y sólo si sus funciones de pertenencia son iguales.

$$\mu_A(x) = \mu_B(x)$$

Tipos de funciones de pertenencia o membresía.

Las funciones de pertenencia o membresía pueden ser definidas prácticamente de cualquier forma. Para la definición de estas funciones, suelen utilizarse convencionalmente ciertas familias de formas estándar para coincidir con el significado lingüístico de las etiquetas más utilizadas.

Existen funciones que por su simplicidad matemática y computacional son usadas frecuentemente en los sistemas difusos, algunas de ellas son: trapezoidal, triangular, S, gamma, campana gaussiana, etc.

Función de pertenencia o membresía trapezoidal

Se define como:

$$A = \begin{cases} 0 & si\,(x \le a)\ o\ (x \ge d) \\ (x-a)/(b-a) & si\ x \in (a,b] \\ 1 & si\ x \in (b,c) \\ (d-x)/(d-c) & si\ x \in (b,d) \end{cases}$$

Y su gráfica característica se muestra en la Figura 2.3:

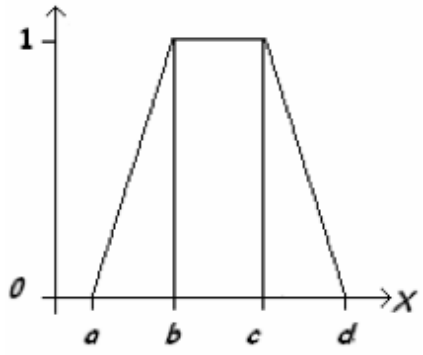

Figura 2.3 Función de pertenencia o membresía trapezoidal

Se utiliza básicamente para sistemas difusos sencillos ya que es posible definir un conjunto difuso con pocos datos y calcular el valor de pertenencia con pocas operaciones. Por estas características, se emplea en sistemas basados en un microprocesador.

Función de pertenencia o membresía triangular

Se define como:

$$A = \begin{cases} 0 & si\ x \le a \\ (x-a)/(m-a) & si\ x \in (a,m] \\ (b-x)/(b-m) & si\ x \in (m,b) \\ 0 & si\ x \ge b \end{cases}$$

Y su gráfica característica se muestra en la Figura 2.4:

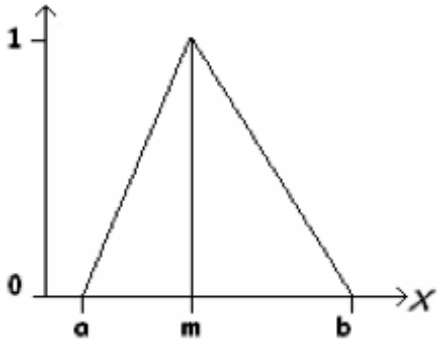

Figura 2.4 Función de pertenencia o membresía triangular

Esta función es adecuada para modelar propiedades con un valor de pertenencia distinto de cero para un rango de valores estrecho entorno a un punto *m*.

Función de pertenencia o membresía S

Se define como:

$$A = \begin{cases} 0 & si\ x \le a \\ 2[(x-a)/(b-a)]^2 & si\ x \in (a,m] \\ 1-2[(x-b)/(b-a)]^2 & si\ x \in (m,b) \\ 1 & si\ x \ge b \end{cases}$$

Y su gráfica característica se muestra en la Figura 2.5:

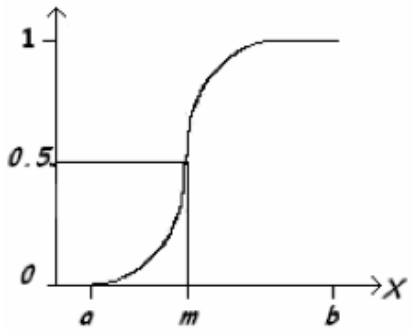

Figura 2.5 Función de pertenencia o membresía S

Esta función resulta benéfica para modelar propiedades como *grande, mucho, positivo*, etc. Se caracteriza por tener un valor de pertenencia distinto de cero para un rango de valores por encima de cierto punta *a*, siendo 0 por debajo de *a* y 1 para valores mayores de *b*. Su punto de cruce es *m=(a+b)/2*; y entre los puntos *a* y *b* es de tipo cuadrático.

Función de pertenencia o membresía Gamma

Está definida como:

$$A = \begin{cases} 0 & si\ x \le a \\ 1 - e^{-k(x-a)^2} & si > a \end{cases}$$

donde k>0.

La gráfica característica se muestra en la Figura 2.6:

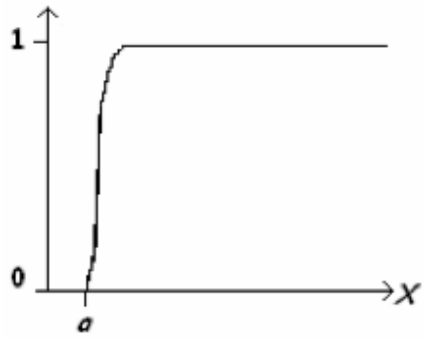

Figura 2.6 Función de pertenencia o membresía Gamma

Este tipo de función se caracteriza por su rápido crecimiento en un corto periodo. Se usa principalmente para modelar las variables lingüísticas extremas como *bebé* o *anciano*.

Función de pertenencia o membresía Gaussiana

Se define como:

$$A = e^{-k(x-m)^2}$$

donde k>0

La gráfica característica se muestra en la Figura 2.7:

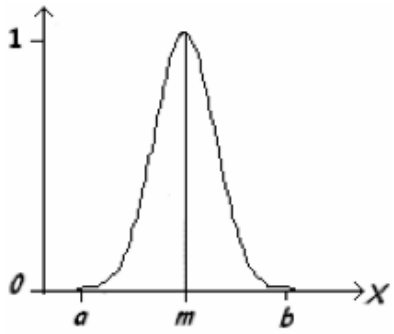

Figura 2.7 Función de pertenencia o membresía Gaussiana

Esta función tiene forma de campana, resulta adecuada para los conjuntos definidos en torno al valor *m* como *medio, normal, cero*, etc. Puede definirse también usando expresiones analíticas exponenciales o cuadráticas como la misma campana de Gauss.

Variable lingüística.

Un conjunto difuso se encuentra asociado a un valor lingüístico que está definido por una palabra, un adjetivo o una etiqueta lingüística.

Una variable lingüística es aquel concepto que será valuado de forma difusa y que se le puede aplicar un adjetivo que define sus características mediante el lenguaje natural. Algunos ejemplos: La temperatura, la edad, la velocidad.

Dada esta definición, se puede llamar valor lingüístico a las diversas formas de clasificar a la variable lingüística. Por ejemplo, para la variable "*temperatura",* los valores podrían ser: "muy caliente", "caliente", "templado", "frío" y "muy frío".

Una *variable difusa* o lingüística es una variable cuyos valores se pueden considerar etiquetas de los conjuntos difusos.

En la Figura 2.8, se muestra que cada valor lingüístico tiene un conjunto difuso asociado. Con esto, se pude hablar de los conjuntos difusos “Bajo”, “Mediado” y “Alto” asociados a la variable lingüística “Altura”.

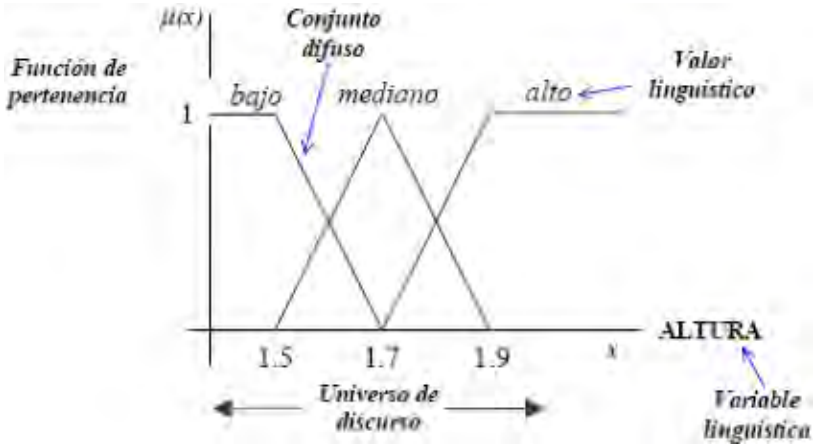

Figura 2.8 Variable lingüística y sus valores lingüísticos asociados a conjuntos difusos

Operaciones sobre los conjuntos difusos.

A los conjuntos difusos se les puede aplicar ciertos operadores o pueden realizarse operaciones entre ellos. Cuando se aplica un operador sobre un conjunto difuso, se obtiene otro conjunto difuso, de forma análoga sucede al combinar dos o más conjuntos difusos con una operación.

Sean los conjuntos difusos *A* y *B* asociados a una variable lingüística *x*, pueden definirse tres operaciones básicas: complemento, intersección y unión:

• Complemento

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

• Unión

$$\mu_{A \cup B}(x) = \max[\mu_A, \mu_B]$$

• Intersección

$$\mu_{A \cap B}(x) = \min[\mu_A, \mu_B]$$

Estas tres operaciones cumplen con la asociatividad, conmutatividad y distributividad como en la teoría de de conjuntos clásica. Existen dos leyes de la teoría clásica de conjuntos que no se cumplen en la teoría de conjuntos difusos, El Principio de Contradicción: $A \cup \bar{A} = U$ y el Principio de de exclusión: $A \cap \bar{A} = \varnothing$. Una de las formas para describir en qué se diferencian la teoría clásica de conjuntos con respecto a la teoría de conjuntos difusos es justamente el

incumplimiento de estas dos leyes. A continuación se presentan algunas propiedades que sólo se cumplen en los conjuntos difusos:

$$U \cap \bar{U} \neq \varnothing$$

$$U \cup \bar{U} \neq N$$

$$A \cap \varnothing = \varnothing \quad \text{ó} \quad \mu_A \wedge 0 = 0$$

$$A \cup \varnothing = A \quad \text{ó} \quad \mu_A \vee 0 = \mu_A$$

$$A \cap N = A \quad \text{ó} \quad \mu_A \wedge 1 = \mu_A$$

$$A \cup N = N \quad \text{ó} \quad \mu_A \vee 1 = 1$$

Donde *N* es el conjunto unidad y ∅ es el conjunto vacío.

Las funciones que corresponden a la unión y a la intersección pueden generalizarse siempre y cuando se cumplan ciertas restricciones, dichas funciones son conocidas como Norma triangular (t-norma) y Conorma triangular (t-conorma). Algunos de los operadores que cumplen con las condiciones para ser Normas y Conormas son:

Normas	Conormas
min(a,b)	max(a,b)
(a+b-ab)	(ab)
a * b = MAX(0,a+b-1)	a + b = MIN(1,a+b)

Tabla 2.1 Tabla con algunos operadores que son normas o conormas

Otra forma de definir los operadores *min* y *max* es la siguiente:

$$a \wedge b = min(a,b) = a \text{ si } a \leq b$$
$$a \wedge b = min(a,b) = b \text{ si } a > b$$

$$a \vee b = max(a,b) = a \text{ si } a \geq b$$
$$a \vee b = max(a,b) = b \text{ si } a < b$$

Como hemos visto, es posible asociar a los operadores difusos *min y max* con las conectivas *AND* y *OR.*

Relación difusa.

Una relación difusa representa el grado de presencia o ausencia de asociación o interconexión entre elementos de dos o más conjuntos difusos. Para dos universos de discurso *U* y *V,* la relación difusa *R(U,V)* es un conjunto difuso en el espacio producto *UxV* que tiene como característica la función de pertenencia $\mu_R(x,y)$ donde "*x*" pertenece a *U* y "*y*" pertenece a *V*, es decir:

$$R(U,V) = \{((x,y), \mu_R(x,y))|(x,y) \in UxV\}$$

donde $\mu_R(x.y) \in [0,1]$.

Como las relaciones difusas son también un conjunto difuso, las operaciones y operadores definidos en la sección anterior pueden aplicarse a éstas. Para *R(x,y)* y *S(x,y)*, que son dos relaciones en el mismo espacio producto *UxV*, la intersección y la unión entre *R* y *S* quedan definidas como:

$$\mu_{R \cap S}(x,y) = \mu_R(x,y) * \mu_S(x,y)$$
$$\mu_{R \cup S}(x,y) = \mu_R(x,y) \oplus \mu_S(x,y)$$

donde * es cualquier t-norma y $\oplus$ es cualquier t-conorma.
Ahora bien, si consideramos que las relaciones difusas *R* y *S* pertenecen a diferentes espacios producto *R(U,V)* y *S(V,W),* su composición difusa se define de forma parecida a la composición clásica teniendo en cuenta que la relación difusa *R* tiene una función característica $\mu_R(x,y)$ y la relación difusa *S* tiene una función característica $\mu_S(y,z)$. Entonces la composición difusa $R \circ S$ se define como una relación difusa *UxW* cuya función de pertenencia es:

$$\mu_{R \circ S}(u,w) = sup_{v \in V}[\mu_R(x,y) * \mu_S(y,z)]$$

donde el operador *sup* es el máximo y el operador * puede ser cualquier t-norma.

Dependiendo de la elección para el operador *, se pueden tener diversos casos particulares de relaciones difusas. Las más habituales son la sup-min (operador "MIN") y la sup-producto (operador producto).

En otras palabras, la composición *sup* de dos relaciones difusas es un conjunto difuso en *UxW.*

Inferencia Difusa.

Como en el caso de la lógica clásica, la lógica difusa también se ocupa del razonamiento formal con proposiciones, pero a diferencia de la clásica, los valores de las proposiciones pueden tomar valores intermedios entre falso y verdadero. De igual manera, como se puede definir un isomorfismo entre la lógica y la teoría clásica de conjuntos, es posible también

definir un isomorfismo entre la lógica y los conjuntos difusos y a su vez, con el álgebra Booleana. De esta forma, los conjuntos difusos también pueden representar predicados en la lógica proposicional.

En la lógica clásica, existen dos importantes reglas de inferencia: *El Modus Ponens Generalizado* y *El Modus Tollens Generalizado*:

• *Modus Ponens Generalizado.*

Es la primer categoría de inferencias, está definido como:

Conocimiento: *Si* x es A *Entonces* y es B
Hecho: x es A'
_____
Consecuencia: y es B'

Donde *A, A', B y B'* son conjuntos difusos.

El objetivo del Modus Ponens Generalizado (GMP) es inferir el efecto dada la causa. Para el caso especial donde A'=A y B'=B entonces el GMP se reduce al Modus Ponens de la lógica ordinaria.

• Modus Tollens Generalizado.

Es la segunda categoría de inferencias, está definido como:

Conocimiento: *Si* x es A *Entonces* y es B
Hecho: y es B'
_____
Consecuencia: x es A'

Donde *A, A', B y B'* son conjuntos difusos.

El objetivo del Modus Tollens Generalizado (GMT) es inferir la causa que conduce a un efecto particular. Para el caso especial en que $A' = \bar{A}$ y
$B' = \bar{B}$ el GTM se vuelve Modus Tollens de la lógica ordinaria.

Implicación Difusa.

En términos de la teoría de lógica difusa, la proposición "Si u es A, entonces v es B", donde A, B y $u \in U$ y $v \in V$, tiene una función característica asociada $\mu_{A \to B}(x,y)$ que toma valores en el intervalo [0,1]. Es decir, cada proposición o regla IF-THEN es un conjunto difuso con su función característica asociada que mide el grado de verdad de la implicación entre "*x*" y "*y*". Algunas de las funciones características asociadas más usadas y resultado de la anterior tautología son:

$$\mu_{A\rightarrow B}(x,y) = 1-\mu_{A\cap B}(x,y) = 1-min[\mu_A(x),1-\mu_B(y)]$$

$$\mu_{A\rightarrow B}(x,y) = max[1-\mu_A(x),\mu_B(y)]$$

Otras funciones características empleadas en la literatura de lógica difusa son:

• Conjunción difusa

$$\mu_{A\rightarrow B}(x,y) = \mu_A(x) * \mu_B(y)$$

• Disyunción difusa

$$\mu_{A\rightarrow B}(x,y) = \mu_A(x) + \mu_B(y)$$

• Modus Ponens Generalizado

$$\mu_{A\rightarrow B}(x,y) = sup\{c\in[0,1]|\mu_A(x) * c \leq \mu_B(y)\}$$

Reglas difusas.

Las reglas difusas combinan uno o más conjuntos difusos de entrada, llamados *antecedentes o premisas*, y se les asocia un conjunto de salida llamado *consecuente o consecuencia.* A las premisas se les puede asociar mediante conectivos lógicos como *y, o,* etc. Una regla típica tiene la forma IF-THEN.

Las reglas difusas permiten expresar el conocimiento acerca de la relación entre antecedentes y consecuentes. Dicho conocimiento no se basa en sólo una regla, normalmente son varias reglas que se pueden agrupar formando una *base de reglas o base de conocimiento.*

Formalmente, una base de reglas difusas está definida como una colección de reglas $R^{(L)}$:

$$R^{(L)}: IF\ x_1\ es\ F_1^L\ ,\ ...\ ,\ x_n\ is\ F_n^L\ THEN\ y\ es\ G^L$$

donde

F_i^L y G^L son conjuntos difusos en $U_i \subset \Re$ y $V \subset \Re$, respectivamente y

$\boldsymbol{x} = [x_1 ... x_n]^T \in U_1 \times U_2 ... U_n$

$y \in V$

Tal que **x**, y son variables lingüísticas.

Este formato de reglas es conocido como *tipo Mandami*. En él, la función de salida es un conjunto difuso.

Existe otro formato que es llamado *tipo Sugeno*. Se caracteriza porque la función de salida es una combinación lineal de las variables de entrada.

$$R^{(L)}: IF\ x_1\ es\ F_1{}^L\ ,\ ...\ ,\ x_n\ es\ F_n{}^L\ THEN\ y^L = f^L(x)$$

En ambos casos, nombramos *S* al número de reglas IF-ELSE de la base de reglas, entonces L= 1,2, ... ,S. El vector ***x*** representa el conjunto de entradas mientras que "*y*" representa la salida del sistema difuso.

Los sistemas difusos con x_n entradas y una salida "*y*", son llamados *MISO(Multiple Input Single Output)* y los sistemas difusos con múltiples entradas y múltiples salidas (de 1 hasta *k*) son conocidos como *MIMO (Multiple Input Multiple Output)*.

Una forma típica usada en el *control difuso* para una base de conocimiento está representada de la forma:

$R^{(L)}$: IF A_1 THEN B_1
OR
IF A_2 THEN B_2
OR
.
.
.
IF A_N THEN B_N

Donde A_1, A_2, ... , A_n son conjuntos difusos de *U* y B_1, B_2, ... , B_n son conjuntos difusos de V, correspondientes a los antecedentes y consecuentes respectivamente.

Sistema de Inferencia Difusa o Controlador difuso

La lógica difusa se usa en sistemas de control difuso que usan expresiones ambiguas para generar reglas que controlan al sistema. Estos sistemas de control funcionan de forma muy diferente a los convencionales; usan el conocimiento de un experto para generar una base de conocimiento que dará al sistema la capacidad de elegir sobre ciertas situaciones en el sistema. Los sistemas de control difuso permiten diseñar un conjunto de reglas que utilizaría una persona (experto) para controlar un proceso, y a partir de las reglas generar un control sobre determinadas acciones.

El control difuso puede aplicarse en diversos sistemas, desde los sencillos hasta los muy complejos. La estructura de un sistema de inferencia difusa o controlador difuso se muestra en la Figura 2.9:

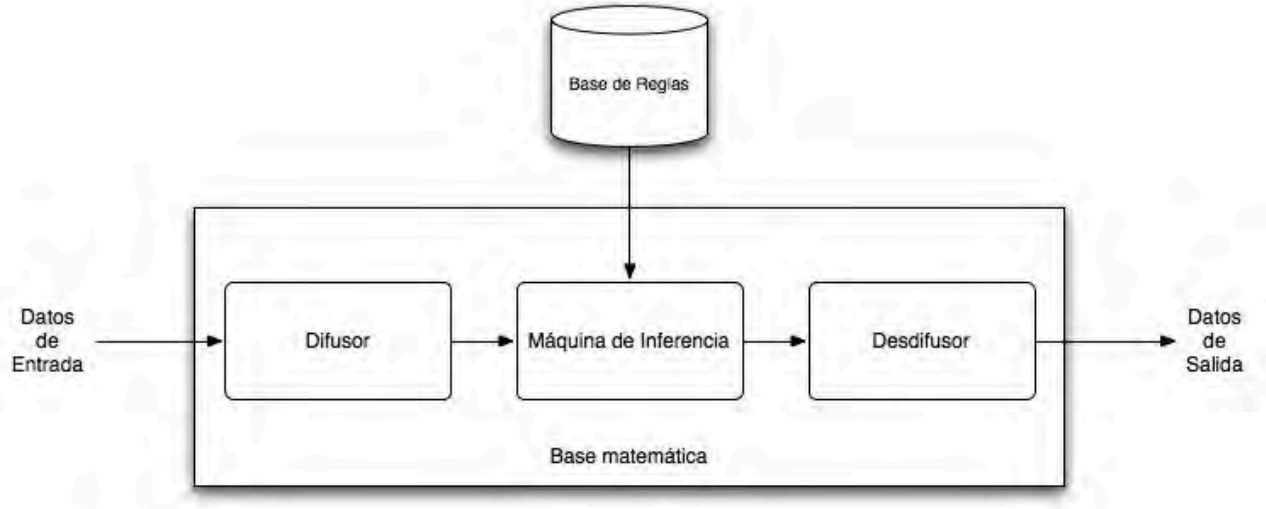

Figura 2.9 Diagrama de un sistema de inferencia difusa o controlador difuso

•*Difusor (Fuzzifier)*

Tiene como objetivo convertir valores reales o concretos en valores difusos, es decir, a las variables de entrada se les asignan grados de pertenencia en relación a los conjuntos difusos previamente definidos usando las funciones de pertenencia asociadas a los conjuntos difusos. Cada conjunto difuso producido por este bloque, está definido sobre el universo del discurso de la variable lingüística respectiva y tiene una función de pertenencia cuya forma puede ser distinta para cada variable de entrada.

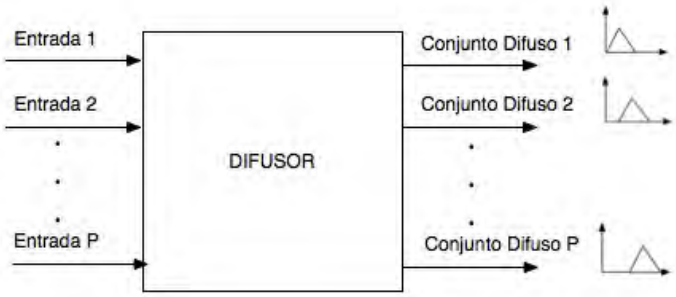

Figura 2.10 Diagrama del bloque difusor

Las entradas de este bloque son valores concretos de las variables de entrada y las salidas son grados de pertenencia a los conjuntos difusos considerados (Figura 2.10).

•*Máquina de inferencia*

Este bloque relaciona los conjuntos difusos de entrada y salida para representar las reglas que definirán al sistema difuso. En la máquina de inferencia, se usa la información de la base de reglas o base de conocimiento para generar reglas mediante el uso de condiciones.

La base de reglas se obtiene para *n* variables de entrada y *m* reglas puede representarse como:

$$Si\ x_1\ es\ A_1\ ,\ \dots\ ,\ x_n\ es\ A_n\ entonces\ \text{“}y\text{”}\ es\ B_m$$

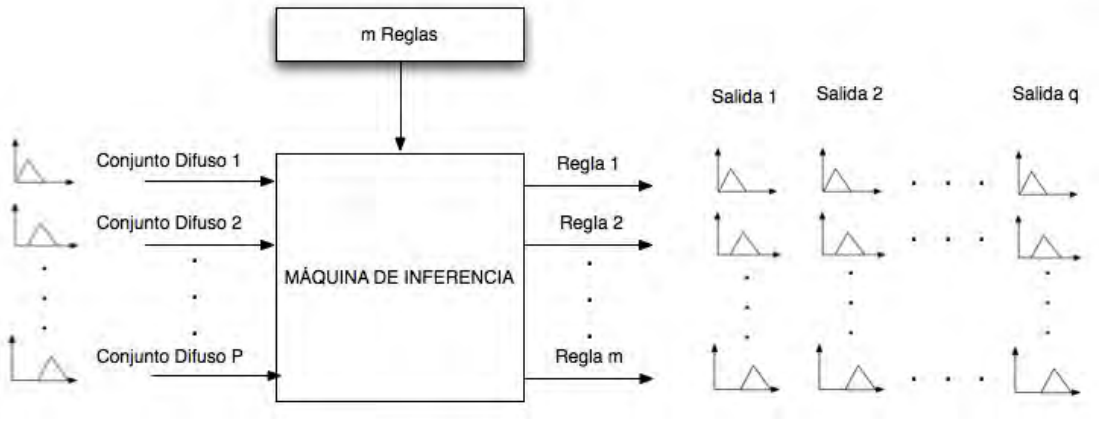

Figura 2.11 Diagrama del bloque de la Máquina de Inferencia

En este bloque, la máquina de inferencia recibe *P* conjuntos difusos producidos por el bloque difusor y los aplica a cada una de las *m* reglas de la base de reglas para producir *m*q* conjuntos difusos, definidos sobre los universos de discurso de las variables lingüísticas de salida.

Las entradas de este bloque son conjuntos difusos (grados de pertenencia) y las salidas son también conjuntos difusos, asociados a las variables de salida (Figura 2.11).

•*Desdifusor (Defuzzifier)*

Este bloque tiene como objetivo obtener un resultado, es decir, un valor real o concreto de la variable de salida a partir del conjunto difuso obtenido de la máquina de inferencia mediante métodos matemáticos de desdifusión.

En general, para producir cada uno de los *q* valores concretos, el difusor toma *m* conjuntos difusos correspondientes a cada variable de salida y mediante algún método o algoritmo, se produce un valor real o concreto (Figura 2.12).

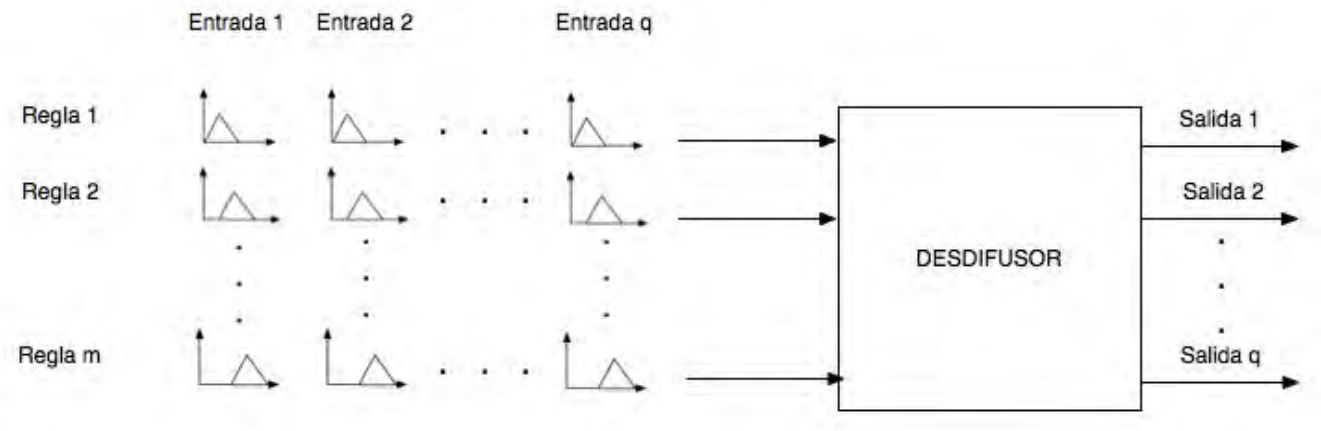

Figura 2.12 Diagrama del bloque del desdifuzor

Existen diversos métodos o algoritmos de desdifusión, los más utilizados son:

- Método del máximo

Se elige como valor para la variable de salida, la función característica del conjunto difuso de salida con el máximo valor. Este método no es óptimo y puede generar demasiados errores ya que el valor máximo puede ser alcanzado por varias salidas.

- Método de altura

Se calcula para cada regla el centro de gravedad del conjunto difuso de salida B_m y después se calcula la salida del sistema como la medida ponderada. Está definido por:

$$y_h = \frac{\int \bar{y}_m \mu_{B_m}(\bar{y}_m) dy}{\int \mu_{B_m}(\bar{y}_m) dy}$$

- Método del centroide

Utiliza como salida del sistema, el centro del área bajo la función de pertenencia que se ha obtenido para los conjuntos difusos de salida. Para obtener el centroide, se utiliza la siguiente ecuación.

$$\hat{y} = \frac{\int y \mu_B(y) dy}{\int \mu_B(y) dy}$$

Donde $\mu_B(y)$ es la función de pertenencia de cada salida obtenida, mientras que la integral es valuada en el intervalo del soporte (*sop(x)*) del conjunto de salida.

Este método es el más utilizado en las aplicaciones de lógica difusa y en la ingeniería ya que con este método se obtiene una solución única.

2.2 Algoritmo de posicionamiento.

Una máquina de estados finitos resulta ideal para diseñar un control de posición que implique rutinas secuenciales de programación, debido a su sencillez de diseño y eficacia al implementarse.

Máquina de estados finitos.

Las máquinas de estados finitos, llamadas también máquinas de estado o autómatas finitos son herramientas muy útiles que sirven para especificar aspectos relacionados con protocolos, arquitecturas de software y computación reactiva, permiten también diseñar circuitos secuenciales complejos capaces de “tomar decisiones” en función de un estado actual del circuito y del valor de sus entradas. En la práctica, se utilizan normalmente como parte del control.

El modelo de la máquina de estado finito, es un modelo que tiene sintaxis y y semántica formales, esto sirve para representar aspectos dinámicos que no se expresan en otros diagramas.

Definición.

La máquina de estado se define como una quíntupla *M* tal que:

$$M = \{E_t, S_t, Q_t, f_n, F\}$$

donde E_t, S_t, Q_t, son conjuntos finitos de las entradas, salidas y estados, respectivamente, y f_n, F son las funciones de salida y de transición de estados.

Tipos de máquinas de estado.

Todas las máquinas de estados finitos tienen un diagrama general, sin embargo, se pueden dividir teniendo en cuenta las posibles modalidades de implementación de la función de salida f_n, en dos clases:

- Máquina de estado tipo Moore
- Máquina de estado tipo Mealy

Máquina de estado tipo Moore

Este tipo de máquinas reciben el nombre de su inventor Edward Forrest Moore. En ellas, las salidas sólo dependen de las variables de estado. De esta forma, las funciones Q_t *y* S_t quedan definidas como:

$$Q_{t+1} = F(E_t, Q_t)$$

$$S_t = f_n(Q_t)$$

Una representación sencilla de esta máquina se ilustra en la Figura 2.13:

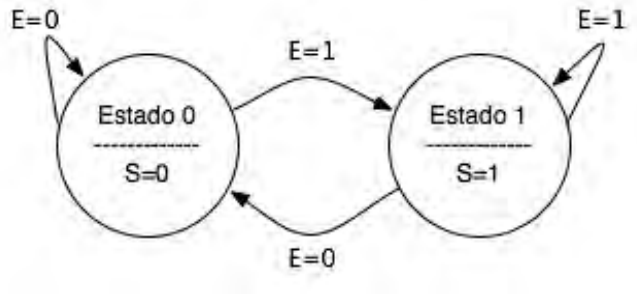

Figura 2.13 Diagrama de una máquina de estado tipo Moore

Máquina de estado tipo Mealy

Al igual que la máquina Moore, la máquina Mealy es denominada así por su creador George H. Mealy. En estas máquinas (quizá las más populares), las salidas están ligadas directamente tanto a las variables de entrada como al estado que presenta. Las funciones Q_t *y* S_t quedan definidas como:

$$Q_{t+1} = F(E_t, Q_t)$$

$$S_t = f_n(E_t, Q_t)$$

Gráficamente, se pude representar a esta máquina como lo muestra la Figura 2.14:

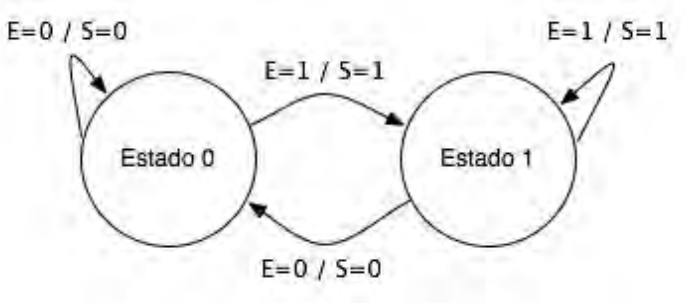

Figura 2.14 Diagrama de una máquina de estado tipo Mealy

A continuación, se presenta una tabla comparativa entre los dos tipos de máquinas.

Moore	Mealy
La salida depende del estado actual.	La salida depende del estado actual y de las entradas.
El número de estados es mayor o igual al número de estados de la máquina Mealy.	Por lo regular, tiene menos o igual número de estados que la máquina Moore.
Las salidas se encuentran dentro del estado.	Las salidas se encuentran fuera del estado, es decir, en la transición.
Es más estable.	Es menos estable.

Tabla 2.2 Tabla comparativa entre los 2 tipos de máquinas de estado

Cabe mencionar que es posible diseñar máquinas de estado híbridas, es decir, cada estado que la componen puede tener las propiedades de una máquina de estados tipo Moore o las propiedades de la máquina de estados tipo Mealy.

Capítulo 3
Entorno de programación.

3.1 LabVIEW.

LabVIEW es un acrónimo de **Lab**oratory **V**irtual **I**nstrument **E**ngineering **W**orkbech. Es un lenguaje y a la vez un entorno de programación gráfica en el que se pueden crear aplicaciones de una forma rápido y sencilla.

Nacional Instruments es la empresa que desarrolla y es propietaria de LabVIEW. Comenzó en 1976 en Austin, Texas; sus primeros productos eran dispositivos para el GPIB, que es un estándar de conexión que permite la comunicación de una computadora con instrumentos electrónicos de medición como osciloscopios o generadores de funciones. En Abril de 1983 comenzó el desarrollo del que sería su producto más sobresaliente: LabVIEW, que fue lanzado en Octubre de 1986 en su versión 1.0 para Macintosh, que en ese entonces eran las computadoras más populares y ya disponían de interfaz gráfica. La versión 2 vio la luz en 1990 y para Windows habría qué esperar a Septiembre de 1992. A continuación, se presenta una tabla en donde se muestra la evolución de LabVIEW.

Fecha	Acontecimiento
Abril de 1983	Comienza el desarrollo de LabVIEW
Octubre de 1986	LabVIEW 1.0 para Macintosh
Enero de 1990	LabVIEW 2.0
Septiembre de 1992	LabVIEW para Windows
Octubre de 1992	LabVIEW para Sun
Octubre de 1993	LabVIEW 3.0 multiplataforma
Mayo 1997	LabVIEW 4.0
Marzo de 1998	LabVIEW 5.0
Febrero de 1999	LabVIEW 5.1 para Linux y Real-Time
Agosto de 2000	LabVIEW 6.0
Mayo de 2003	LabVIEW 7.0, LabVIEW PDA y FPGA
Mayo de 2005	LabVIEW DSP
Junio de 2005	LabVIEW Embedded

Fecha	Acontecimiento
Octubre de 2005	LabVIEW 8.0
Agosto de 2006	LabVIEW 8.2 (Edición especial 20 aniversario)
Abril de 2007	LabVIEW 8.5
Abril de 2009	LabVIEW 9.0 (64 bits, SSL para servicios Web)
Abril de 2010	LabVIEW 10.0 (Combinación de instrucciones)

Tabla 3.1 Cronología de LabVIEW (1983-2010)

LabVIEW es una herramienta de programación gráfica, originalmente estaba enfocada a aplicaciones de control de instrumentos electrónicos usadas en el desarrollo de sistemas de instrumentación, lo que se conoce como instrumentación virtual. Por esta razón, los programas creados en LabVIEW, se guardan en ficheros llamados VI (Virtual Instrument o en español, Instrumento Virtual) y llevan la extensión ".vi".

Aunado a este concepto, se dan nombre a sus dos ventanas principales:

• Panel frontal

• Diagrama de bloques

Panel Frontal

Se trata de la interfaz gráfica con el usuario. Esta interfaz recoge los datos de entrada que el usuario proporciona y también representa los datos de salida que arroja el programa. El panel frontal está constituido por una serie de botones, perillas, potenciómetros, pantallas, etc. Cada elemento puede definirse como un control o un indicador. El primero sirve para introducir datos al VI y los indicadores se emplean para mostrar los resultados producidos, ya sean datos adquiridos o resultados del VI (Figura 3.1).

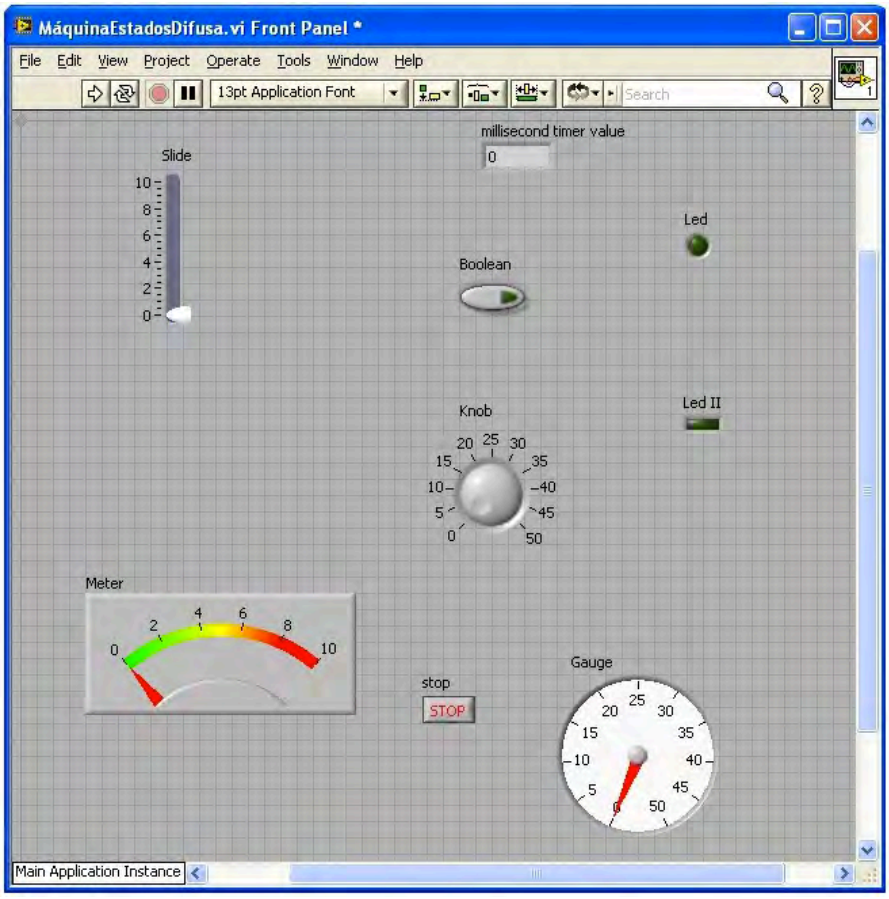

Figura 3.1 Panel Frontal

Diagrama de Bloques

Constituye el código fuente del VI. En él, se realiza la implementación del programa del VI para controlar o realizar un proceso utilizando las señales de entrada y salida que se han creado en el Panel Frontal. El Diagrama de Bloques incluye funciones y estructuras que están integradas en las bibliotecas de LabVIEW. Estas funciones son nodos elementales y son análogas a los operadores o bibliotecas de funciones de los lenguajes habitualmente usados.

Los controladores e indicadores que se colocaron previamente en el Panel Frontal se hacen presentes en el Diagrama de Bloques mediante las terminales (Figura 3.2).

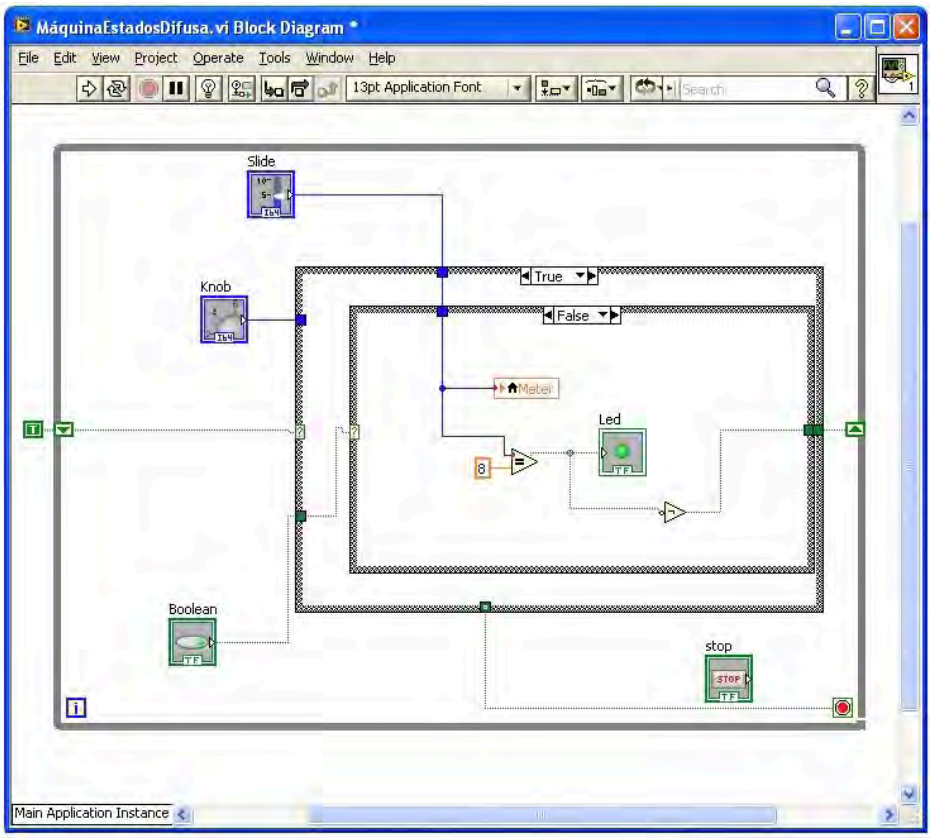

Figura 3.2 Diagrama de Bloques

El Diagrama de Bloques se construye conectando los distintos objetos entre sí, como si de un circuito se tratara. Los cables unen terminales de entrada y de salida con los objetos correspondientes y así es como fluyen los datos.

LabVIEW posee una enorme biblioteca de funciones, entre ellas, matemáticas, aritméticas, comparativas, de conversión, de entrada-salida, de arreglos, de procesamiento de señales, etc. Las estructuras, similares a las declaraciones causales y a los bucles en lenguajes convencionales, ejecutan el código que contienen de forma condicional o repetitiva (*for*, *while*, etc). Los cables son las trayectorias que siguen los datos desde su origen hasta su destino, ya sea una función, una estructura, una terminal, etc. Cada cable tiene un color distinto, dependiendo el tipo de dato que transporta.

En la parte superior de estas dos ventanas, se sitúa una barra con diversas herramientas. En el diagrama de bloques, esta barra tiene algunas opciones más:

i) Controles de ejecución

El primer grupo de herramientas sirve para controlar la ejecución de un programa. El primer botón indica si hay errores en el programa (una flecha rota) o no los hay (flecha completa, como en la ilustración). El segundo programa ejecuta el programa de forma continua. El tercer botón detiene la ejecución y el cuarto permite hacer una pausa en la ejecución del programa (Figura 3.3).

Figura 3.3 Controles de ejecución

ii) Controles de depuración

El segundo bloque de botones sirve para ayudar a la depuración del programa. El primer botón es el *Highlight Execution*, una de las herramientas más útiles para depurar; ejecuta paso a paso el programa, permitiendo ver el camino o flujo de los datos. El siguiente botón, *Retain Wire Values* permite retener el valor actual del cable durante la ejecución del programa. Los tres siguientes sirven para ejecutar el programa paso a paso (Figura 3.4).

Figura 3.4 Controles de depuración

iii) Menú para dar formato a textos

Este menú desplegable permite dar formato a textos que se integren en el Diagrama de Bloques (Figura 3.5).

Figura 3.5 Menú para dar formato a textos

iv) Controles de distribución de los objetos

El siguiente grupo de botones sirve para alinear, distribuir, dimensionar, agrupar y ordenar los objetos que forman el programa (Figura 3.6).

Figura 3.6 Controles de distribución de los objetos

v) Búsqueda y ayuda

La caja de texto sirve para insertar una palabra la cuál será asociada a el contenido de LabVIEW y devolverá la lista de coincidencias en donde aparece la palabra insertada. El siguiente botón, abre la ayuda de LabVIEW. Es muy útil ya que al activar esta opción y al

pasar el cursor por cualquier elemento del Diagrama de Bloques, se desplegará una breve descripción del mismo (Figura 3.7).

Figura 3.7 Búsqueda y ayuda

vi) Icono del Programa o VI

Finalmente, aparece el icono que representa al programa o VI. Tiene un menú que permite acceder a diversas propiedades como la versión del programa, la categoría o la edición del icono (Figura 3.8).

Figura 3.8 Icono del Programa o VI

Para colocar funciones en el Diagrama de Bloques y terminales en el Panel Frontal se tienen menús flotantes llamados paleta de funciones y paleta de controles, respectivamente. Existe también la llamada paleta de herramientas, la cuál sirve para editar los objetos que componen al programa o VI. Las paletas de funciones y de controles se despliegan haciendo click con el botón secundario del cursor sobre una zona vacía del Diagrama de Bloques o del Panel Frontal.

Paleta de herramientas

Contiene una serie de elementos que ayudan a la confección del Diagrama de Bloques y a mejorar la estética del Panel Frontal (conexiones de los elementos mediante cableado, añadir texto para una mejor comprensión de cada parte del programa, verificación de resultados, modificar el color de los botones o pantallas tanto del Panel Frontal como del Diagrama de Bloques, etc). La Paleta de herramientas se muestra en la Figura 3.9:

Figura 3.9 Paleta de herramientas

Paleta de Controles

En esta paleta se pueden seleccionar las terminales que servirán para interactuar con el usuario. Las terminales se dividen en indicadores y controles. Los controles representan las entradas de datos que son proporcionados por el usuario y los indicadores representan a las salidas que arroja el VI. Tanto entradas como salidas pueden configurarse para ser del tipo numérico, booleano, string o un arreglo (Figura 3.10).

Figura 3.10 Paleta de Controles

Paleta de Funciones

En ella se puede acceder a las diferentes funciones, sub VI's y estructuras disponibles. Existen varios submenús que se dividen dependiendo la aplicación. Algunas de las funciones más usadas son las matemáticas, programación, temporización, adquisición y análisis de señales, de entrada-salida de datos a un fichero, etc.

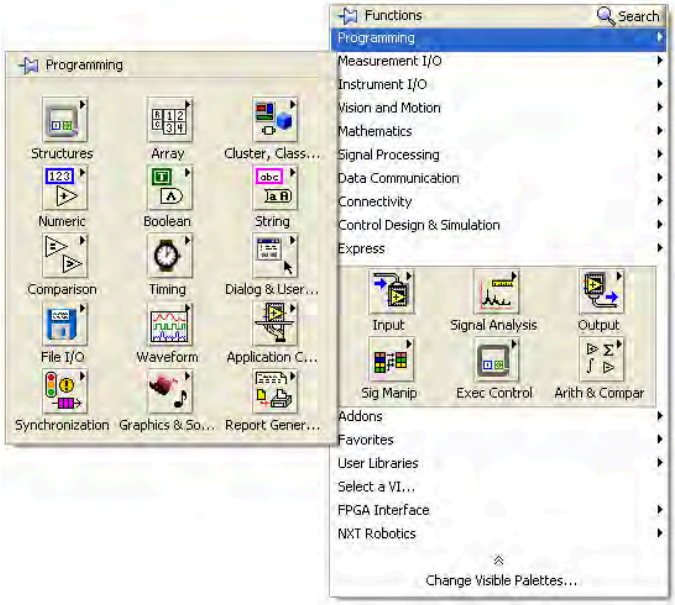

Figura 3.11 Paleta de Funciones

Para poder insertar al Panel Frontal o al Diagrama de Bloques algún elemento contenido en las paletas, sólo basta con arrastrarlo al lugar en dónde se desee colocar (Figura 3.11).

Existen algunas funciones que no vienen incluidas en LabVIEW. La mayoría de estas funciones (o módulos) están diseñados para fines muy específicos, para poder hacer uso de ellos es necesario descargarlas de la página oficial de National Instruments y en algunos casos, pagar por ellos. Entre los más conocidas, están: Control Design and Simulation, Data Comunication, Vision and Motion, FPGA Interface, NXT Robotics.

3.2 Módulo “NI Vision”.

El módulo de LabVIEW *NI Vision Development* está diseñado para el uso de científicos, ingenieros y técnicos que desarrollan aplicaciones de visión artificial e imágenes científicas. LabVIEW soporta gran cantidad de cámaras, ya sean webcams, microscópicas, escáneres y muchas otras más. Algunas bondades de este módulo son:

• *Elección de la cámara*

El software de National Instruments es compatible con cientos de cámaras analógicas de bajo costo hasta cámaras de escaneo de líneas de alta velocidad, incluyendo GigE Vision y cámaras IEEE 1394.

• *Escalabilidad del hardware*

Así como escoger la cámara correcta es importante para cualquier aplicación, la escalabilidad es importante también. Debido a que la tecnología en el desarrollo de cámaras avanza vertiginosamente, algún día se deseará cambiar la cámara por una de mejores características. National Instruments proporciona varios controladores para poder cambiar la cámara sin necesidad de cambiar el software.

• *Algoritmos de amplitud y precisión*

Todas las funciones proporcionadas por el módulo NI Vision Development presentan la ventaja de utilizar precisión de subpixeles para realizar interpolaciones, medir distancias, grados y demás medidas que van desde 1/10 de píxel.

• *Desempeño del algoritmo*

El módulo está altamente optimizado para maximizar el rendimiento de cada posible fuente, resultando ser un software que rivaliza sin problemas con otros paquetes de software de alta velocidad a nivel mundial.

A continuación, se presenta una breve descripción de las funciones que se incluyen en el módulo *NI Vision Development*:

Vision and Motion

Provee una serie de paquetes que permiten la combinación de la visión artificial y la tecnología de control de movimiento (Figura 3.12).

Figura 3.12 Paleta Vision and Motion

NI - IMAQ

Esta paleta es necesaria para establecer y configurar el sistema de adquisición de imágenes. Los VI incluidos en este paquete permiten abrir y cerrar una interfase, configurar la forma de adquisición, etc. (Figura 3.13).

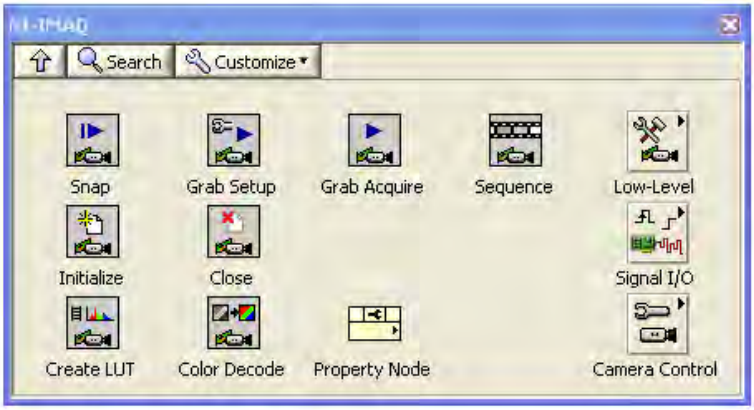

Figura 3.13 Paleta NI - IMAQ

Vision Utilities

Permite crear y manipular imágenes extraídas desde archivos, establecer regiones de interés, calibrar imágenes para realizar conversiones de medidas en pixeles al mundo real, manipular pixeles, sobreponer líneas o rectángulos, etc. (Figura 3.14).

Figura 3.14 Paleta Vision Utilities

Image Processing

Presenta un conjunto de VI de procesamiento de imágenes con el cuál se pueden analizar, filtrar y tratarlas. También es posible hacer un procesamiento de colores, morfología y detectar texturas de la imágenes (Figura 3.15).

Figura 3.15 Paleta Image Processing

Machine Vision

Permite realizar acciones propias de la visión artificial, es decir, inspección, cuantificación y medición de objetos, ausencia o presencia de partes en una imagen, localizar bordes y contornos, medir intensidades, referenciar una imagen con algún sistema de coordenadas, selección dinámica de regiones de interés, encontrar y cuantificar patrones, etc. (Figura 3.16).

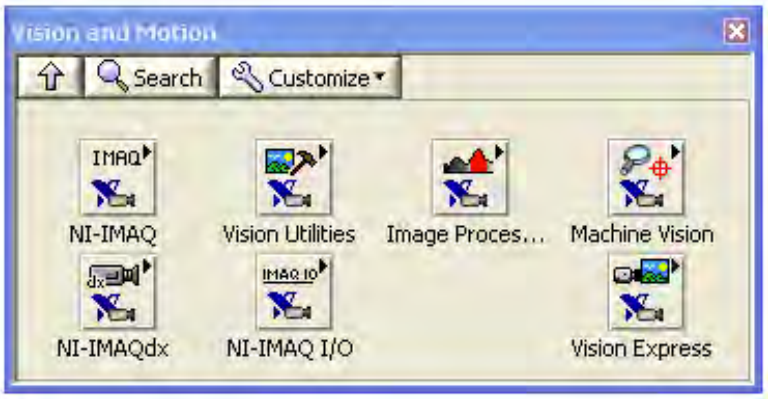

Figura 3.16 Paleta Machine Vision

NI - IMAQdx

Soporta cámaras GigE, IEEE 1934 y especificaciones de instrumentación para cámaras digitales (IIDC). Controla los modos disponibles y las características de las cámaras con las que tiene compatibilidad (Figura 3.17).

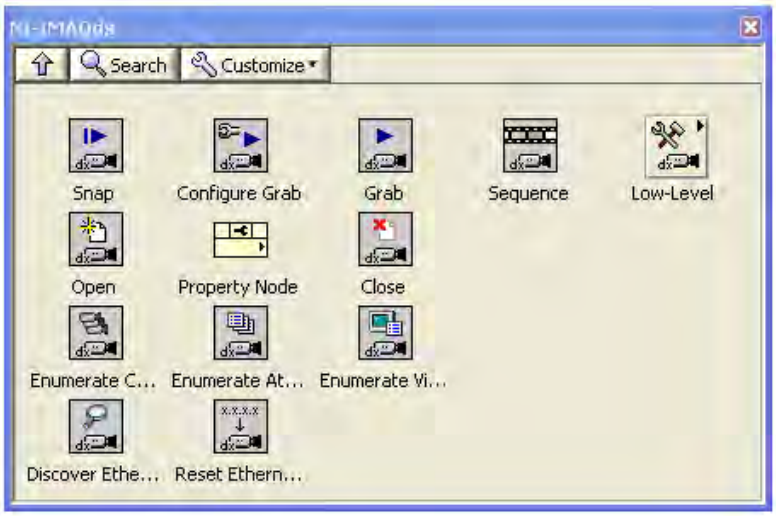

Figura 3.17 Paleta Machine Vision

NI - IMAQ I/O

Esta paleta sirve para crear aplicaciones con las que se puede controlar las líneas de entrada - salida en un dispositivo compatible con las funciones *NI - IMAQ* (Figura 3.18).

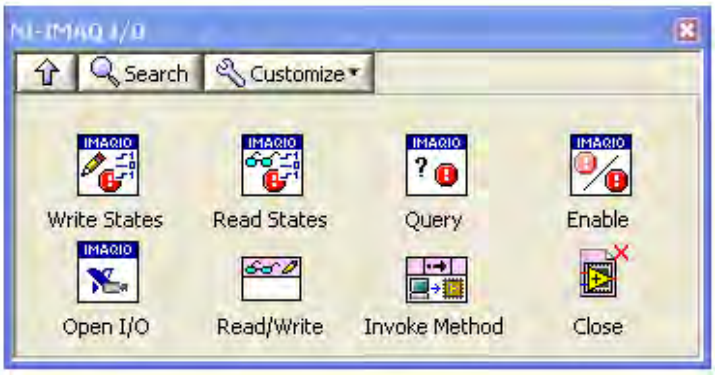

Figura 3.18 Paleta NI - IMAQ I/O

Vision Express

Permite desarrollar de una forma rápida y sencilla la adquisición y el procesamiento de imágenes a través de dos aplicaciones (Figura 3.19):

- *Vision Adquisition*.- Ayuda a configurar fácilmente cámaras analógicas, digitales, IEEE 1934, GigE, etc.

- *Vision Assistant*.- Permite hacer las tareas de procesamiento de imágenes.

Figura 3.19 Paleta Vision Express

Algo que resulta primordial en el uso y adquisición de imágenes es la gestión y manejo de la memoria. En el paquete *Vision Utilities*, se encuentra la paleta de funciones *Image Management* que permite crear, eliminar, definir parámetros, permitir la lectura y escritura, crear copias, etc, de una imagen (Figura 3.20).

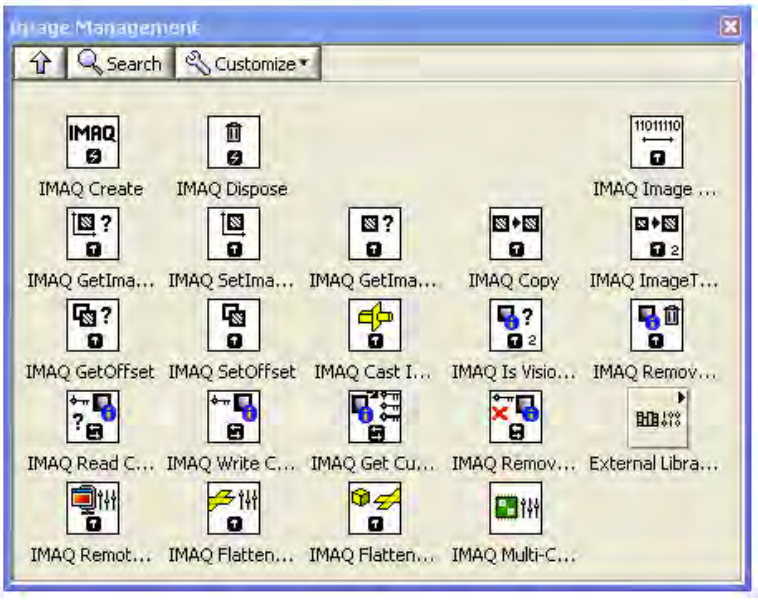

Figura 3.20 Paleta Image Management

Para que LabVIEW pueda manejar archivos gráficos o capturas, debe reservar espacio en memoria para su almacenamiento temporal. Algunas de las funciones más importantes de esta paleta son:

• IMAQ Create

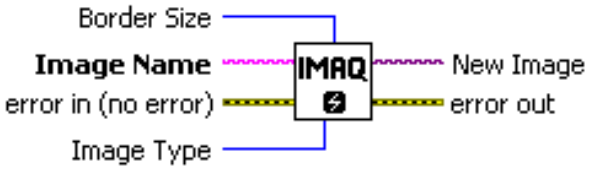

Figura 3.21 IMAQ Create

Donde *Border Size* (tamaño de borde) determina el tamaño en pixeles del borde a crear alrededor de la imagen; los bordes de una imagen nunca son mostrados o almacenados en un archivo. *Image name* (nombre de la imagen) es el nombre asociado a la imagen creada; cada imagen creada debe tener un nombre único. *Error in* describe el estado de error antes de ejecutar el VI, por defecto, no existe error. *Image Type* (tipo de imagen) especifica el tipo de imagen que puede ser: escala de grises, compleja, de colores, etc. *New Image* (imagen nueva) es la salida de la imagen que es utilizada como fuente de diversas funciones usadas posteriormente; múltiples imágenes pueden crearse en una aplicación de LabVIEW. Finalmente, *error out* describe el estado del error después de ejecutar el VI (Figura 3.21).

• IMAQ Dispose

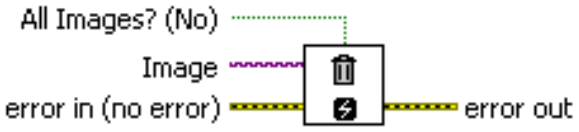

Figura 3.22 IMAQ Dispose

Así como es importante reservar localidades de memoria, también es importante liberar espacio en la memoria cuando ya no se utilizará más, este VI es de gran utilidad para esta tarea. *All images?* (todas las imágenes?) especifica si hay qué destruir una imagen o todas las imágenes antes creadas; al dar un valor booleano verdadero, destruye todas las imágenes antes creadas, de lo contrario, se eliminará sólo la imagen que se tiene como entrada. *Image* (imagen) recibe la imagen que será destruida. *Error in* y *error out* tienen la misma función que en *IMAQ Create* (Figura 3.22).

• IMAQ Copy

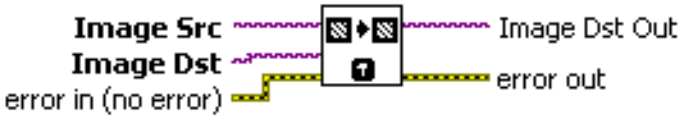

Figura 3.23 IMAQ Copy

Crea una copia de toda la información relacionada con la imagen: borde, pixeles, información de calibración, etc, en otra imagen del mismo tipo. Se usa generalmente con el fin de guardar una copia original de una imagen antes del tratamiento de la misma. *Image Src* (imagen fuente) hace referencia a la imagen fuente. *Image Dst* (imagen destino) hace referencia a la imagen destino. *Error in* y *error out* tienen la misma función que en *IMAQ Create* (Figura 3.23).

3.3 Módulo “Fuzzy Logic”.

El módulo *Fuzzy Logic* no es un módulo que esté aislado; está contenido en el módulo *NI Control Desing and Simulation*, que fue lanzado en la primavera del 2004. LabVIEW amplió sus capacidades para el control de sistemas, para el diseño y análisis de sistemas dinámicos de una forma considerable. Este paquete de funciones es equiparable al *Control System Toolbox* de Matlab (Figura 3.24).

Figura 3.24 Paleta Control Desing and Simulation

El módulo *Fuzzy Logic* resulta ser una excelente plataforma para diseñar, implementar y probar sistemas de control basadas en técnicas de lógica difusa. Este módulo incluye la edición de variables difusas, múltiples formas de funciones de membresía, un amigable editor de reglas *if-then* asociado al control difuso y muchos métodos de desdifusión como el método del centroide, centro del máximo, etc. (Figura 3.25).

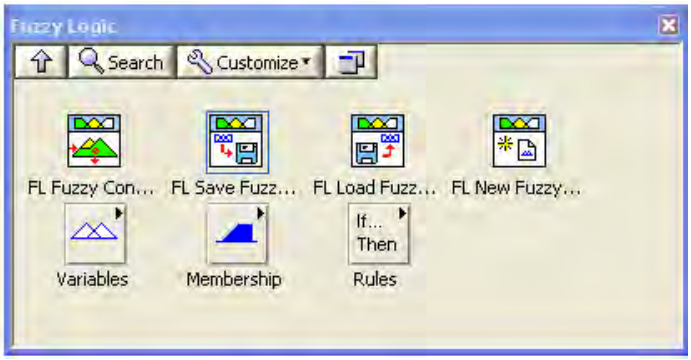

Figura 3.25 Paleta Fuzzy Logic

FL Fuzzy Controller

Implementa un controlador difuso para un sistema difuso que se especifique. Puede configurarse como:

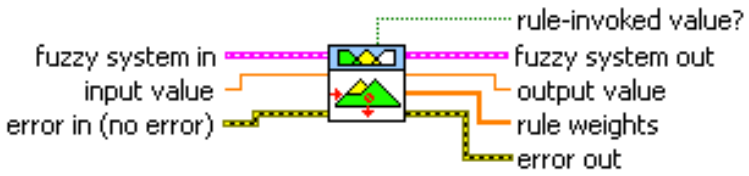

Figura 3.26 VI Fuzzy Controller

Fuzzy system in (Entrada del sistema difuso) especifica la información completa del sistema difuso. *Input value* (valor de entrada) es el valor de entrada al sistema difuso. *Error in* describe el estado de error antes de ejecutar el VI, por defecto, no existe error. *Rule-invoked*

value? (valor para regla invocada?) indica si el controlador difuso ha invocado alguna regla para evaluar el correspondiente valor de salida; cuando el valor es verdadero, no invoca regla alguna, de lo contrario, invoca alguna regla de la base de conocimiento. *Fuzzy system out* (salida del sistema difuso) devuelve la información completa del sistema difuso. *Output value* (valor de salida) devuelve el valor de salida del sistema difuso. *Rule weights* (peso de las reglas) devuelve el peso de las reglas que el controlador difuso usa para escalar las funciones de membresía de las variables lingüísticas de salida. *Error out* describe el estado del error después de ejecutar el VI (Figura 3.26).

FL Save Fuzzy System

Salva un sistema difuso en un archivo .fs, se usa comunmente en conjunto con el VI *Load Fuzzy System.*

Figura 3.27 VI Save Fuzzy System

Fuzzy system in (entrada del sistema difuso) especifica la información completa del sistema difuso. *File path* (ruta del archivo) especifica la ruta del archivo .fs, debe especificarse una ruta absoluta. *Error in* describe el estado de error antes de ejecutar el VI. *Fuzzy system out* (salida del sistema difuso) devuelve la información completa del sistema difuso. *Error out* describe el estado del error después de ejecutar el VI (Figura 3.27).

FL Load Fuzzy System

Carga un sistema difuso desde un archivo .fs, puede usarse el VI anterior para salvar el archivo después de cualquier modificación realizada al sistema difuso.

Figura 3.28 VI Load Fuzzy System

Donde *file path* (ruta del archivo) especifica la ruta en dónde está guardado el archivo .fs, *error in* describe el estado de error antes de ejecutar el VI, *fuzzy system out (salida del sistema difuso) devuelve la información completa del sistema difuso, error out* describe el estado del error después de ejecutar el VI (Figura 3.28).

FL New Fuzzy System

Con este VI se puede crear un sistema difuso, sin embargo, puede construirse con el *Fuzzy Sistem Designer*.

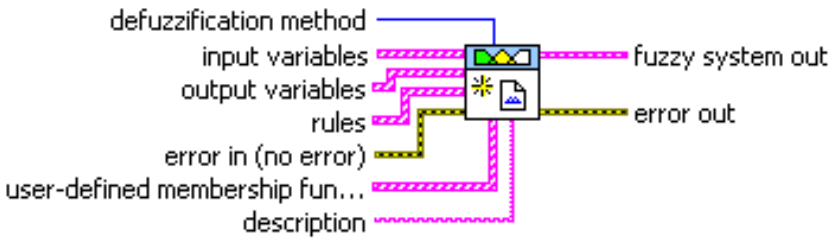

Figura 3.29 VI New Fuzzy System

Deduzzification method (métodos de desdifusión) especifica los métodos de desdifusión que usa el VI para convertir el grado de pertenencia de las funciones de membresía de las variables de salida en valores numéricos. *Input variables* (variables de entrada) especifica las variables lingüísticas de entrada del sistema difuso. *Output variables* (variables de salida) especifica las variables lingüísticas de salida del sistema difuso. *Rules* (reglas) especifica las reglas del sistema difuso, las variables de entrada y de salida son los antecedentes y consecuentes, respectivamente. *E*rror out describe el estado del error después de ejecutar el VI. *User-defined membership function shapes* (formas de las funciones de membresía definidas por el usuario) especifica los valores de los pares ordenados x,y que definen las formas de las funciones de membresía; la forma de las funciones de membresía determina el grado de pertenencia de una variable lingüística que corresponde a un término lingüístico. *Description* (descripción) permite realizar una descripción del sistema difuso. *Fuzzy system out (salida del sistema difuso) devuelve la información completa del sistema difuso.* Error out describe el estado del error después de ejecutar el VI (Figura 3.29).

Variables

Esta paleta permite modificar las variables lingüísticas en los sistemas difusos. Las variables lingüísticas representan, en palabras, las variables de entradas y las de salida del sistema de control (Figura 3.30).

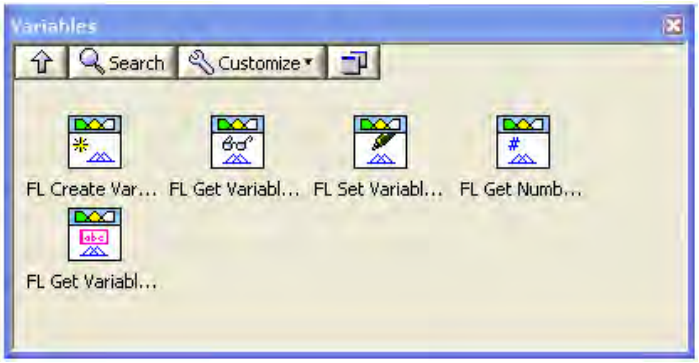

Figura 3.30 Paleta Variables

Membership

Permite modificar las funciones de membresía asociadas a las variables lingüísticas en un sistema difuso. Las funciones de membresía representan el grado de pertenencia de las variables lingüísticas en términos del lenguaje natural o palabras. El grado de pertenencia va de 0 a 1 (Figura 3.31).

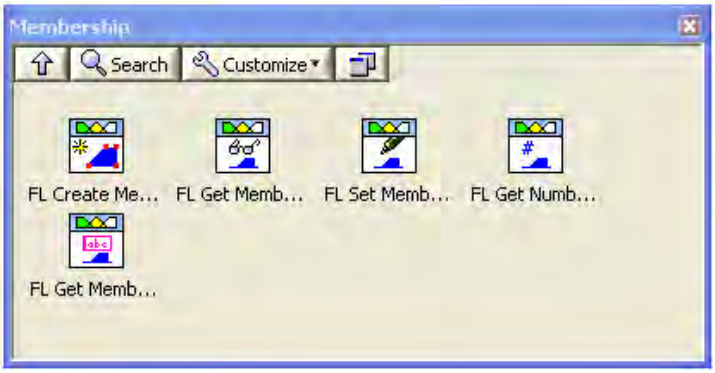

Figura 3.31 Paleta Membership

Rules

Permite modificar las reglas que rigen al sistema difuso. Las reglas describen en palabras, las relaciones entre entradas y salidas en términos del lenguaje natural o palabras (Figura 3.32).

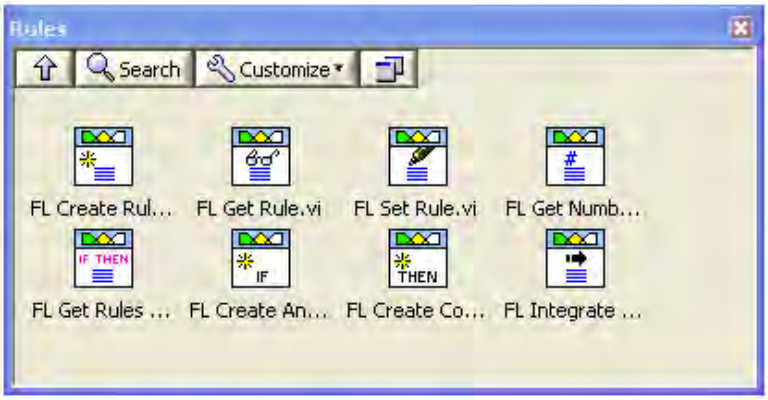

Figura 3.32 Paleta Rules

El módulo de *Fuzzy Logic* ofrece entre sus funciones un asistente para diseñar sistemas difusos de una forma sencilla e intuitiva. Su nombre es *Fuzzy System Designer*. Está compuesto por tres secciones:

• Editor de variables

• Base de reglas

• Prueba del sistema

Editor de variables

Permite crear, nombrar y editar las funciones de membresía asociadas a cada una de las variables lingüísticas de entrada y de salida. Es aquí en donde se pueden definir el número y las formas de las funciones de membresía asociadas a las mismas (Figura 3.33).

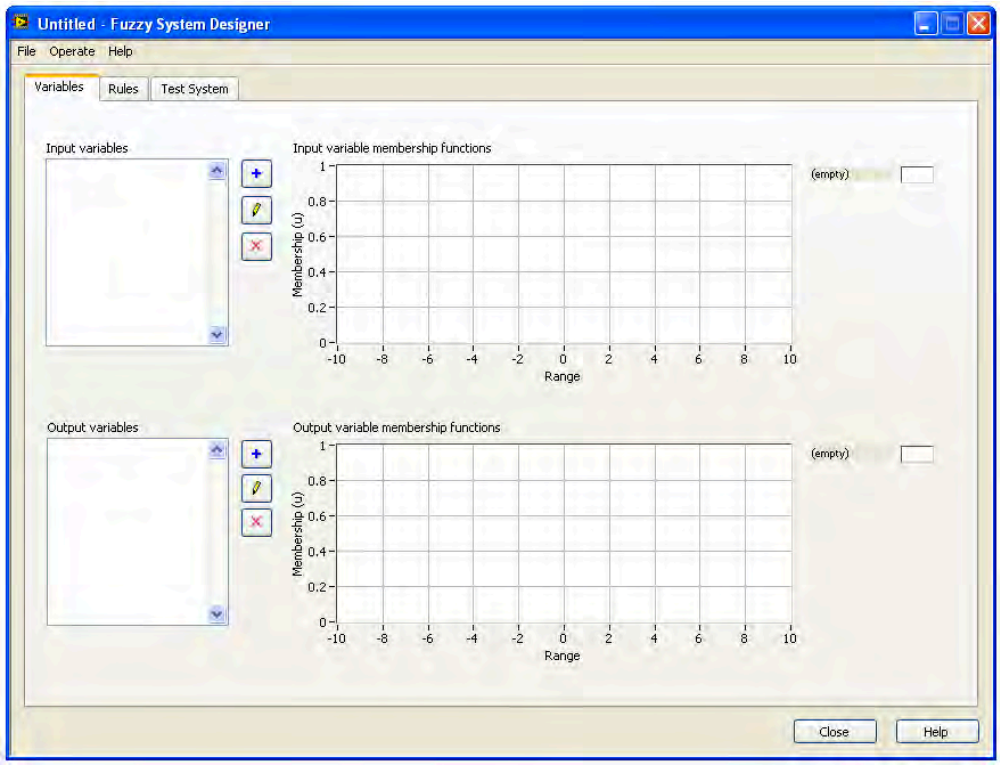

Figura 3.33 Editor de variables

Base de reglas

Permite ingresar la base de reglas, cuya dimensión está determinada por la cantidad de variables y términos lingüísticos de entrada. Cada regla está asociada a un factor de peso (grado de pertenencia) para acentuar o reducir la influencia de una determinada regla en el controlador difuso. En esta pestaña, se elige el método de desdifusión para el sistema difuso. Los métodos que podemos eligir son: centro del área, centro del área modificada, centro de sumas, centro del máximo, media del máximo (Figura 3.34).

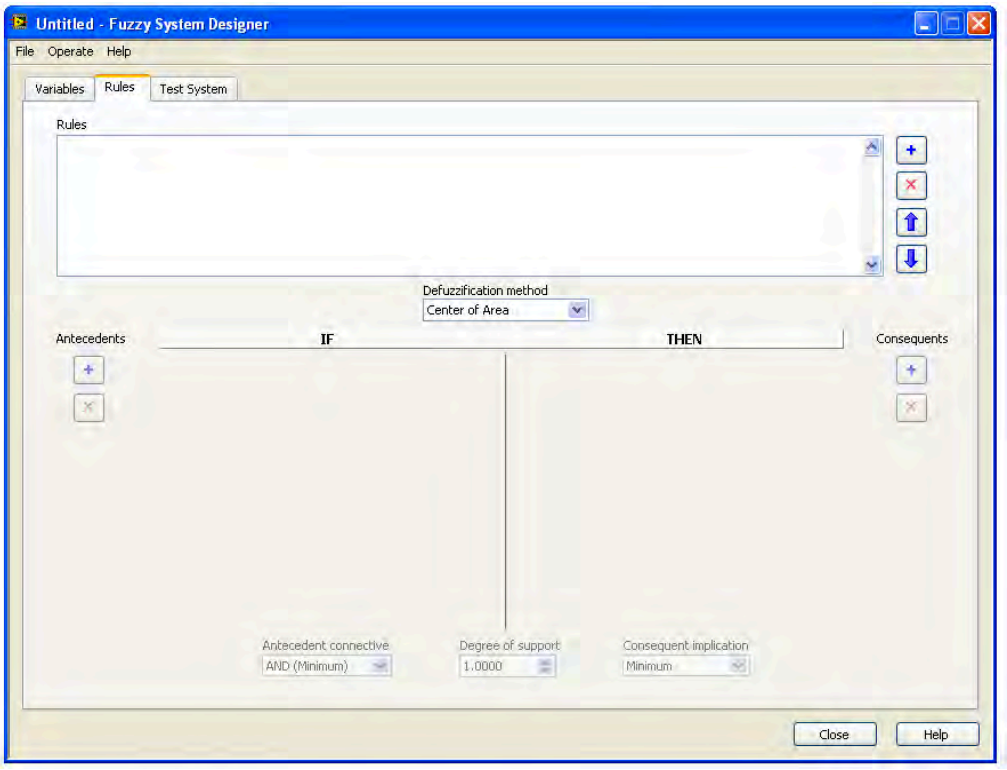

Figura 3.34 Base de reglas

Sistema de prueba

Es la parte final del asistente. En esta sección se puede comprobar el comportamiento del sistema difuso, es decir, la relación que hay entre los valores de entrada y los de salida para validar la base de reglas o conocimiento. En otras palabras, es una simulación del nuevo sistema difuso creado (Figura 3.35).

El resultado de estas pruebas es mostrado en una gráfica.

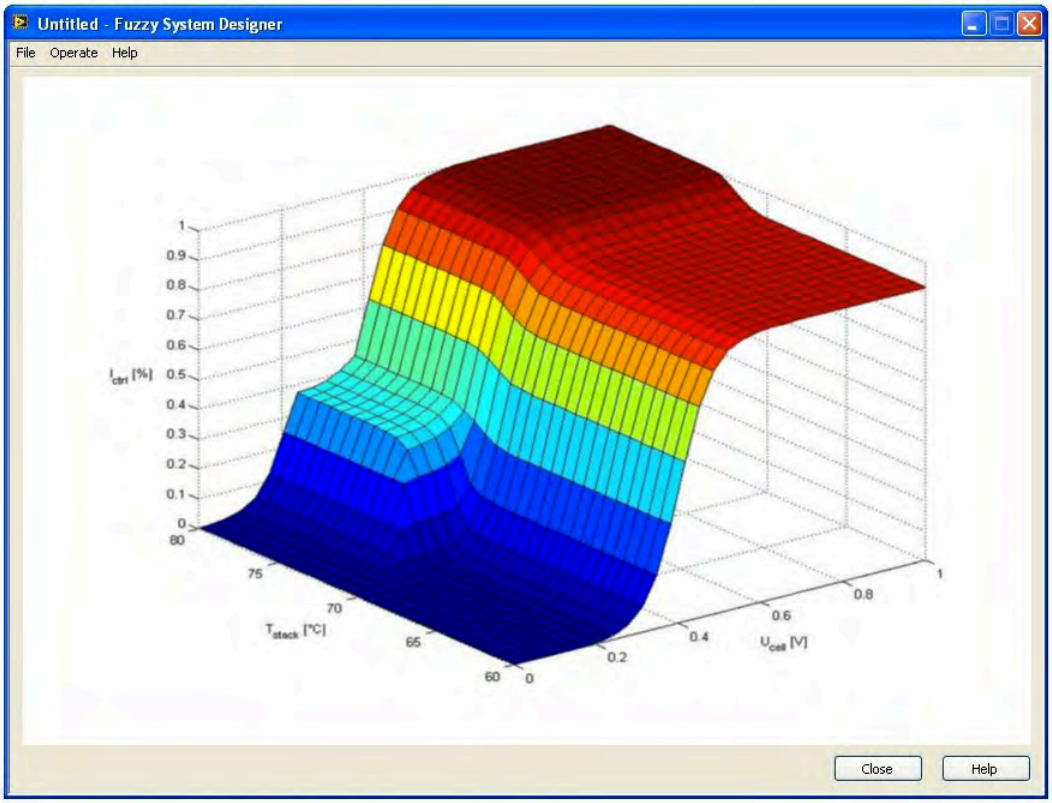

Figura 3.35 Sistema de prueba

3.4 Módulo “NI NXT Robotics”.

El módulo *NI NXT Robotics* permite utilizar las herramientas de programación de LabVIEW para el control del bloque NXT, superando las limitaciones del entorno de programación propio de Lego Mindstorms. Ofrece más libertad de control y expande los límites en el desarrollo de proyectos más complejos (Figura 3.36). Algunas de las bondades que ofrece este módulo son:

• Compilar y descargar un programa elaborado con LabVIEW al NXT. Durante la ejecución del programa es posible interactuar con él. Agregando un control, LabVIEW puede enviar datos al robot y condicionar su comportamiento, agregando un indicador, es posible tomar valores en un determinado punto de la ejecución del programa y enviarlos a LabVIEW para su procesamiento.

• Escribir un programa que se ejecute en la computadora y se comunique con el bloque NXT a través de USB o Bluetooth.

• Si el usuario es desarrollador de un nuevo sensor o de nuevos componentes de hardware, LabVIEW permitirá la creación de bloques nativos para la programación y el control del hardware creado para su uso en el propio entorno.

Figura 3.36 Paleta NXT Robotics

Ahora, se hará una breve descripción de las funciones que contiene este módulo:

NXT Programming

Permite usar los VI básicos de programación que incluye LabVIEW para usarse en conjunto con los VI propios de Lego Mindstorms. Incluye ciclos, varios tipos de datos y funciones para manipular datos (Figura 3.37).

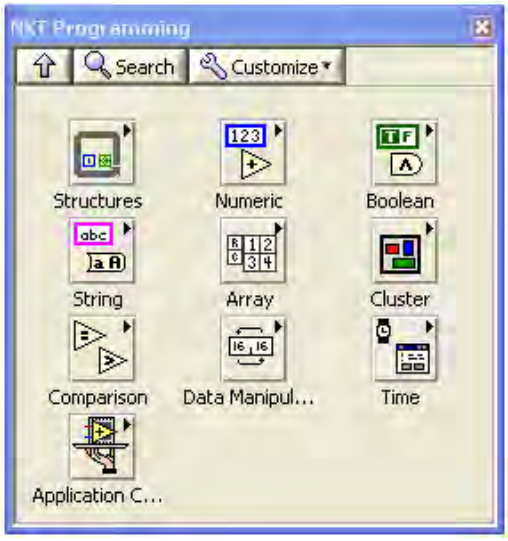

Figura 3.37 Paleta NXT Programming

NXT I/O

Esta paleta nos ofrece funciones para programar el hardware, es decir, servomotores, sensores, display, sonidos, funciones del Bluetooth, etc. También nos ofrece las funciones de temporización y configuración de la transmisión de datos, así como configuraciones más avanzadas de hardware (Figura 3.38).

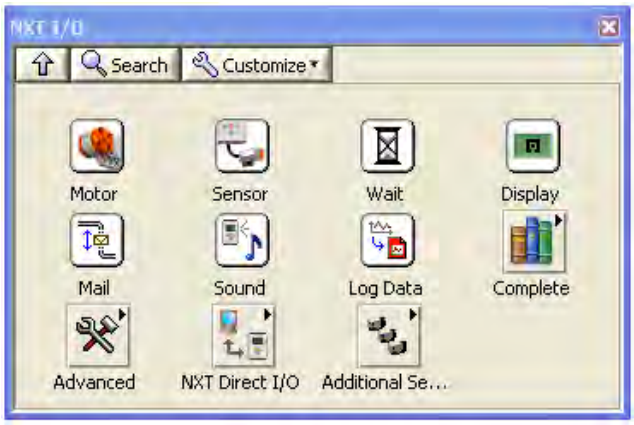

Figura 3.38 Paleta NXT Programming

Behaviors

Las funciones que ofrece esta paleta son VI preprogramados para hacer que el robot NXT presente varias maniobras. En estos VI están involucrados los servomotores y algunos sensores (Figura 3.39).

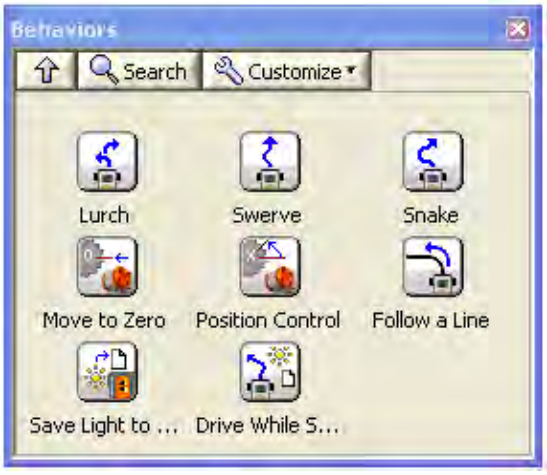

Figura 3.39 Paleta Behaviors

TETRIX

Este conjunto de funciones permite manipular un robot usando sensores de datos o controladores tales como enconders o joysticks (Figura 3.40). Permite también el uso de motores alimentados con corriente directa (DC).

Figura 3.40 Paleta TETRIX

Algunos de los VI más usados dentro de este módulo por el hardware que controlan son:

• Motor On

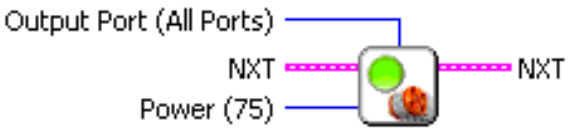

Figura 3.41 VI Motor On

Enciende el motor conectado en el puerto especificado con cierto nivel de potencia. *Outout Port* (Puerto de salida) es el puerto conectado al motor, por defecto, están activados todos los puertos. *NXT* conecta con un previo o posterior VI para generar un flujo en el programa. *Power* (Potencia) es el nivel de potencia que va desde -100 a 100 y el valor por defecto es 75 (Figura 3.41).

• Motor Reverse

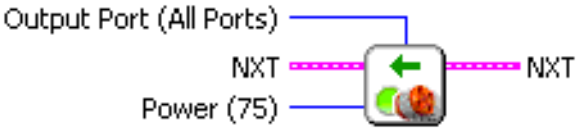

Figura 3.42 VI Motor Reverse

Enciende el motor conectado en el puerto especificado pero en sentido inverso del nivel de potencia de entrada, lo que provoca que se mueva en sentido inverso. *Outout Port* (Puerto de salida) es el puerto conectado al motor, por defecto, están activados todos los puertos. *NXT* conecta con un previo o posterior VI para generar un flujo en el programa. *Power* (Potencia) es el nivel de potencia que va desde -100 a 100 y el valor por defecto es 75 (Figura 3.42).

• Motor Brake

Figura 3.43 VI Motor Reverse

Detiene el motor conectado en el puerto especificado. *Outout Port* (Puerto de salida) es el puerto conectado al motor, por defecto, están activados todos los puertos. *NXT* conecta con un previo o posterior VI para generar un flujo en el programa (Figura 3.43).

• Read Rotation

Figura 3.44 VI Read Rotation

Lee el sensor de rotación del motor conectado en el puerto especificado y regresa el valor en grados. *Outout Port* (Puerto de salida) es el puerto conectado al motor, por defecto, está activado el puerto A. *NXT* conecta con un previo o posterior VI para generar un flujo en el programa. *Angle of rotation* (Ángulo de rotación) devuelve el valor del sensor de rotación del motor en grados (Figura 3.44).

• Read Touch

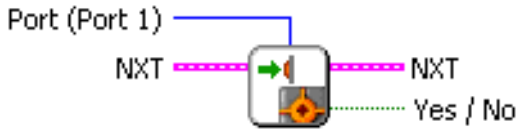

Figura 3.45 VI Read Touch

Lee el sensor de toque conectado en el puerto especificado; la salida es un valor booleano. *Port* (Puerto) es el puerto conectado el sensor de toque, por defecto, está activado el puerto 1. *NXT* conecta con un previo o posterior VI para generar un flujo en el programa. *Yes/No* (Sí/No) devuelve “yes” si está presionado el sensor, y “no” si no está presionado (Figura 3.45).

• Read Light

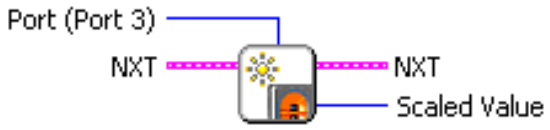

Figura 3.46 VI Read Light

Lee el sensor de luz con el led integrado encendido, conectado en el puerto especificado. *Port* (Puerto) es el puerto conectado el sensor de luz, por defecto, está activado el puerto 3. *NXT* conecta con un previo o posterior VI para generar un flujo en el programa. *Scaled Value* (Valor a escala) devuelve el valor del sensor que va de 0 a 100 (Figura 3.46).

• Read Ultrasound

Figura 3.47 VI Read Ultrasound

Lee el sensor ultrasónico conectado en el puerto específico y devuelve el valor en centímetros. *Port* (Puerto) es el puerto conectado el sensor ultrasónico, por defecto, está activado el puerto 4. *NXT* conecta con un previo o posterior VI para generar un flujo en el programa. *Distance* (Distancia) devuelve el valor del sensor en centímetros (Figura 3.47).

• Wait for Time

Figura 3.48 VI Wait for Time

Espera el número de segundos especificados. *NXT* conecta con un previo o posterior VI para generar un flujo en el programa. *Time* (Tiempo) es el tiempo que debe esperar, está definido en segundos (Figura 3.48).

• Play Piano File

Figura 3.49 VI Wait for Time

Reproduce un archivo específico (.pno) de una melodía en piano. *NXT* conecta con un previo o posterior VI para generar un flujo en el programa. *Song* (Canción) es el nombre del archivo a reproducir, por omisión, el archivo a reproducir es song.pno, cargado en el bloque NXT (Figura 3.49).

Capítulo 4
Desarrollo del sistema de control.

4.1 Planteamiento del problema.

El problema del control de robots móviles por medio de la visión artificial y reconocimiento de patrones ha sido tratado desde diversas perspectivas y contextos; sin embargo en la gran mayoría de los casos, los conocimientos de programación y de procesamiento digital de imágenes necesarios para hacer la implementación de los sistemas, suelen ser complejos, dificultando la investigación, experimentación y desarrollo de aplicaciones que puedan hacer uso de esta tecnología.

El trabajo que ahora se presenta, tiene como objetivo, el desarrollo de un sencillo sistema que permita controlar un robot móvil que recolecte cierta cantidad de objetos en un espacio delimitado por el alcance visual de un dispositivo de adquisición de imágenes, usando como método el reconocimiento de patrones (Figura 4.1).

Figura 4.1 Sistema diseñado

4.2 Descripción del escenario de operación y del robot móvil.

Adquisición de imágenes

Por la naturaleza del sistema propuesto, el dispositivo de adquisición de imágenes, necesita estar a una distancia constante y sin movimiento alguno, es decir, estático. Para llevar a cabo esta condición, se ha construido con tubos de P.V.C. de 2 pulgadas de diámetro, un soporte con forma de pirámide truncada en cuyo centro de la base superior está situado el dispositivo de adquisición de imágenes y está a una altura de 185.5 [cm] con respecto al nivel del suelo. Esta altura fue seleccionada de forma arbitraria y sirvió como constante para posteriores cálculos (Figura 4.2).

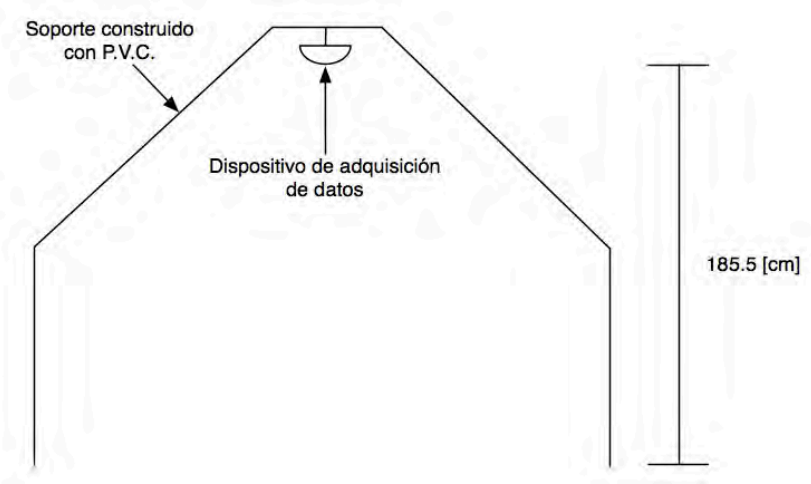

Figura 4.2 Soporte para el dispositivo de adquisición de datos

Cámara web

Se ha seleccionado una cámara web como dispositivo de adquisición de imágenes. La cámara web es una *Logitech HD Webcam C270* (Figura 4.3) y tiene las siguientes especificaciones:

• Captura de imágenes: Hasta 3 megapixeles.

• Captura de video: Máximo 1280 x 720 pixeles en modo HD.

• Cuadros por segundo (FPS): Hasta 30.

• Micrófono integrado.

• Interfaz: USB 2.0.

Figura 4.3 Cámara web Logitech C270

Esta cámara web fue elegida por el bajo costo que tiene con respecto a otras cámaras web de similares características, por tener una velocidad de adquisición de imágenes de hasta 30 frames por segundo (FPS) y por ser compatible con DirectShow, una API y framework multimedia creada por Windows y que LabVIEW usa como bus de comunicación para dispositivos de adquisición de imágenes.

LabVIEW permite configurar diversos parámetros de la cámara y entre ellos está la posibilidad de configurar las dimensiones de las imágenes que adquirirá. Se han elegido 1024 x 576 pixeles como dimensiones de la imagen a adquirir debido a que el tiempo de procesamiento es más breve con respecto a una imagen que tenga el tamaño máximo (1280 x 720 pixeles) y el tamaño elegido no difiere en forma abrupta con el tamaño máximo.

A la altura y configuración especificadas, la cámara tiene una visión rectangular que tiene las siguientes medidas: 165 [cm] de largo y 94 [cm] de ancho. Estas medidas determinan el área de acción del robot móvil (Figura 4.4).

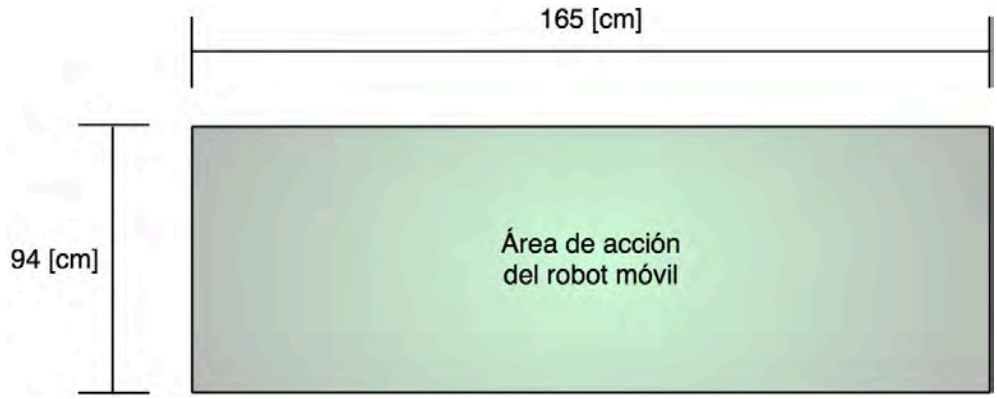

Figura 4.4 Área de acción del robot móvil

Si a esta área de acción la referenciamos con el sistema de coordenadas que LabVIEW usa para el procesamiento de la imagen, queda de la forma mostrada en la Figura 4.5:

Figura 4.5 Área de acción del robot móvil

Robot móvil

La construcción del robot móvil se realizó en su totalidad con piezas del kit Lego Mindstorms. La configuración que se ha elegido es la de pistas de deslizamiento ya que permite un control simple, la respuesta en terrenos irregulares y a campo abierto es muy aceptable, la resistencia al desgaste es grande y la adherencia a la superficie de apoyo es mayor, reduciendo la tendencia al derrapamiento (Figura 4.6).

Figura 4.6 Robot móvil construido

El robot móvil consta de dos partes principales: el sistema de locomoción y el sistema de sujeción.

• Sistema de locomoción

Está constituido por dos servomotores, cada servomotor gobierna el comportamiento de una pista de deslizamiento u oruga. Los motores son controlados de forma independiente por el microcontrolador, de esta forma, se garantiza el control total del desplazamiento del robot móvil (Figuras 4.7a y 4.7b).

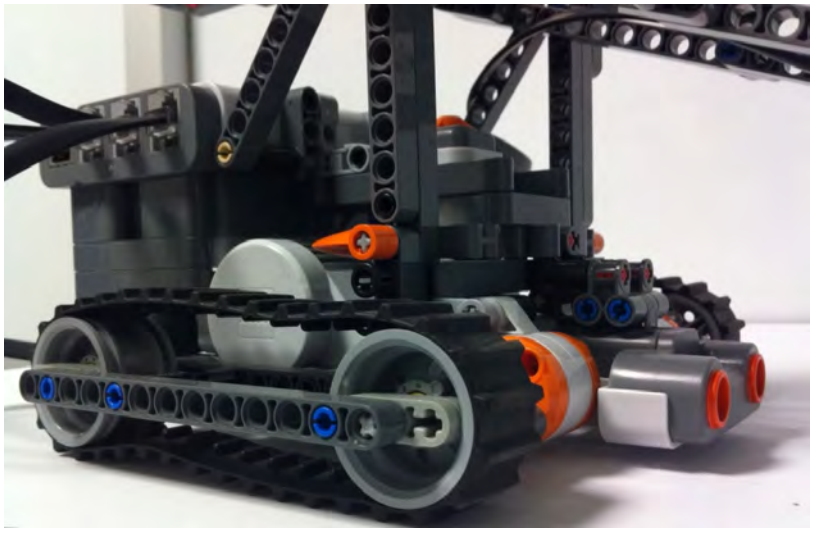

Figura 4.7a Sistema de locomoción

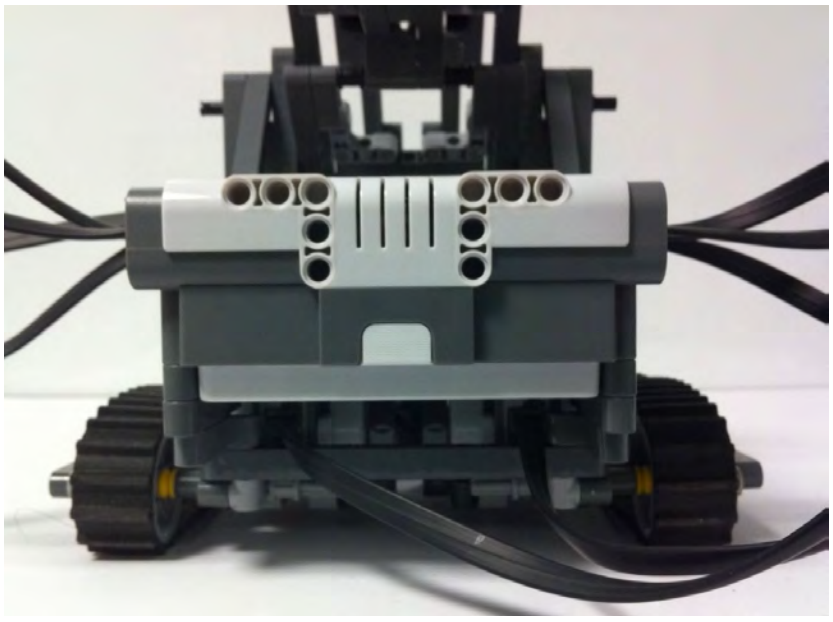

Figura 4.7b Vista trasera del sistema de locomoción

• Sistema de sujeción

Un servomotor gobierna el comportamiento de esta parte del robot móvil. Tiene un brazo rígido que termina en una garra de 2 dedos con la cuál se sujetan los objetos que recolectará el robot móvil. Cuando el servomotor gira en sentido antihorario, el brazo se inclina de tal forma que la garra queda a la altura de la superficie de apoyo y la garra se abre; esta es la posición en la cuál el robot móvil sujetará el objeto. Cuando el servomotor gira en sentido horario, la garra se cierra y el brazo se incorpora; este movimiento supone que el objeto está ya en la garra (Figuras 4.8a, 4.8b y 4.8c).

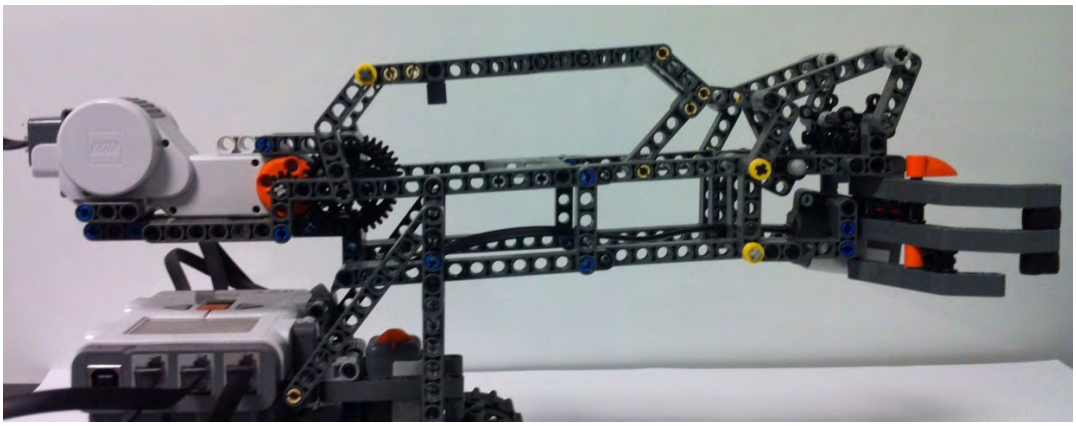

Figura 4.8a Vista lateral del sistema de sujeción

Figura 4.8b Vista trasera del sistema de sujeción

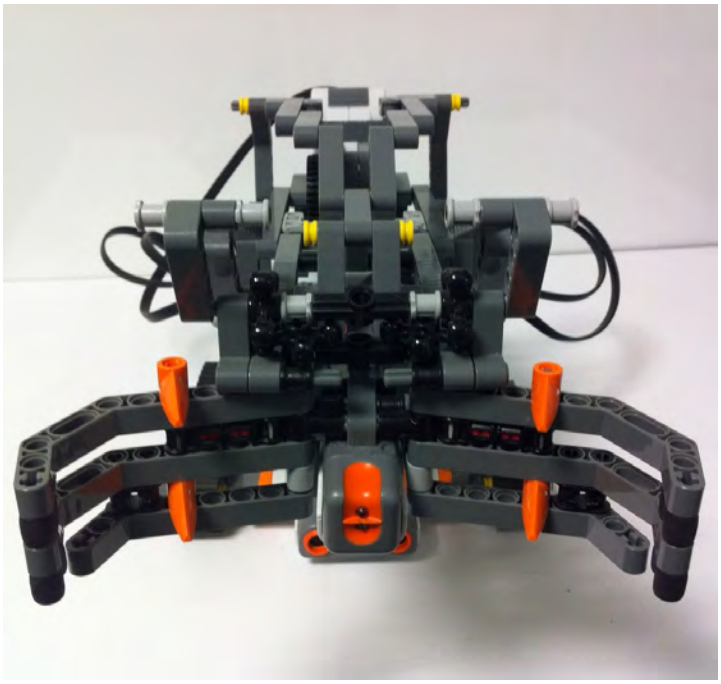

Figura 4.8c Vista frontal del sistema de sujeción

4.3 Diseño del controlador difuso.

Se ha elegido un controlador difuso de tipo *Mandami* para este sistema, es decir, tiene como función de salida un conjunto difuso. Como se ha visto en el capítulo dos, este tipo de sistema tiene valores numéricos de entrada que se difusificarán para obtener conjuntos difusos que con la ayuda de una máquina de inferencia y una base de reglas o conocimiento obtendrá un conjunto difuso de salida que se desdifusificará para finalmente obtener valores numéricos de salida, que son valores manejables por el resto de los procesos involucrados en el control del robot móvil.

Variables de entrada

El desplazamiento del robot móvil esencialmente consiste en dos movimientos: traslación y rotación sobre el mismo eje. En primera instancia, el robot móvil debe alinearse con el objeto y para eso necesita girar cierta cantidad de grados para llevarlo a cabo. Ya alineado, el robot móvil está a cierta distancia la cuál debe recorrer para llegar al objeto y poder sujetarlo. Cuando el robot móvil tiene el objeto sujeto, debe hacer el alineamiento y recorrer cierta distancia con respecto al punto de referencia (origen).

Dados estos movimientos, se han asignado cuatro variables de entrada: “*distanciaObjeto*”, “*anguloDeGiroObjeto*”, “*distanciaOrigen*”, “*anguloDeGiroOrigen*”. Cada una de estas variables tiene funciones de membresía asociadas y cierto rango de operación que está

limitado por el área de acción y la cantidad de grados que debe girar. A continuación se describe cada variable de entrada:

• *distanciaObjeto*

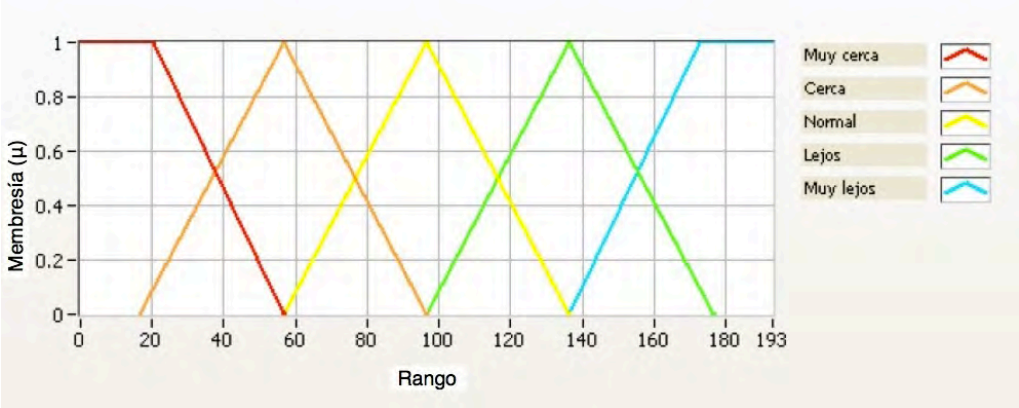

Figura 4.9 Variable de entrada "distanciaObjeto"

Esta variable tiene cinco funciones de membresía asociadas: "Muy cerca" que es una función trapezoidal, "Cerca", "Normal", "Lejos", que son funciones triangulares y "Muy lejos", función trapezoidal (Figura 4.9). El rango de operación de esta variable es de 0 a 193 [cm]. El valor máximo fue calculado tomando en cuenta las dimensiones en centímetros del área de acción, es decir, x=165, y=94; tomando estos valores para dibujar un triángulo rectángulo y obtener la hipotenusa h (Figura 4.10).

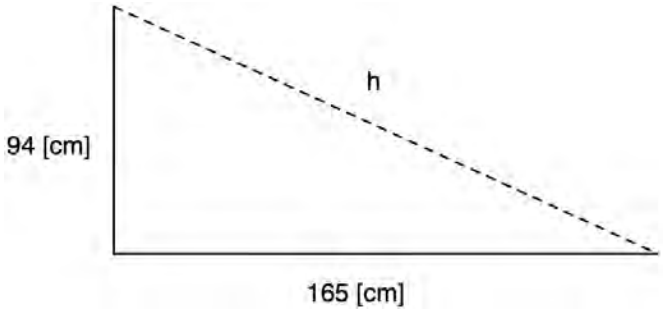

Figura 4.10 Triángulo construido con las dimensiones reales del área de acción del robot móvil

Por el teorema de Pitágoras, es posible obtener el valor de *h*:

$$h^2 = a^2 + b^2$$

$$h = \sqrt{a^2 + b^2}$$

Si se asignan valores a las variables *a*=165 y *b*=94:

$$h = \sqrt{165^2 + 94^2}$$
$$h = \sqrt{27225 + 8836}$$
$$h = \sqrt{36061}$$
$$h = 189.8973 \approx 190$$
$$h = 190[cm]$$

Considerando que los cálculos de la distancia entre el objeto y el robot móvil se hacen con respecto al centroide de éste y dando 3 [cm] como rango de error a causa del tiempo de procesamiento de la imagen, el valor máximo para la variable de entrada "*distanciaObjeto*" es:

$$h_T = 190 + 3$$
$$h_T = 193[cm]$$

•*anguloDeGiroObjeto*

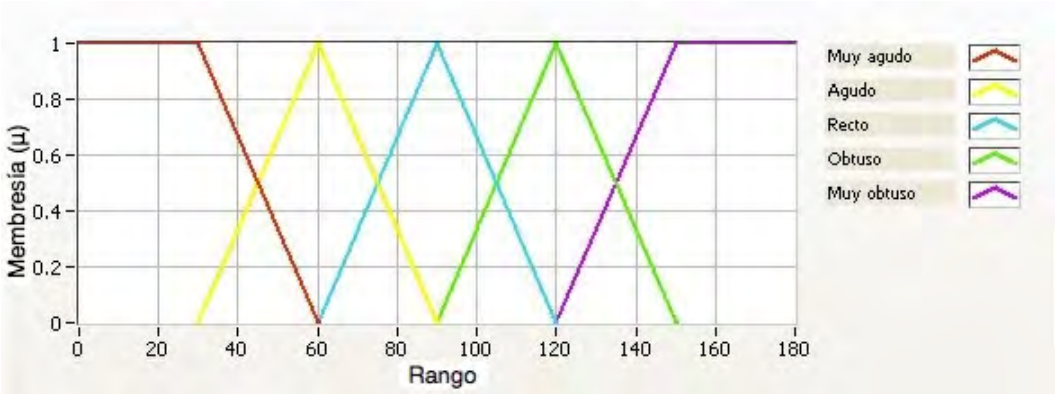

Figura 4.11 Variable de entrada "anguloDeGiroObjeto"

Esta variable cuenta con cinco funciones de membresía asociadas: "Muy agudo", que es una función trapezoidal, "Agudo", "Recto", "Obtuso", que son funciones triangulares y "Muy obtuso" que es una función trapezoidal (Figura 4.11). El rango de operación de esta variable es de 0º a 180º. Se ha considerado este rango de valores debido a que se ha desarrollado un algoritmo capaz de decidir hacia qué dirección debe rotar el robot móvil, con el propósito de girar lo menos posible y ahorrar energía. De esta forma, se asegura que la cantidad de grados que debe girar, siempre será menor o igual a 180º.

• *distanciaOrigen*

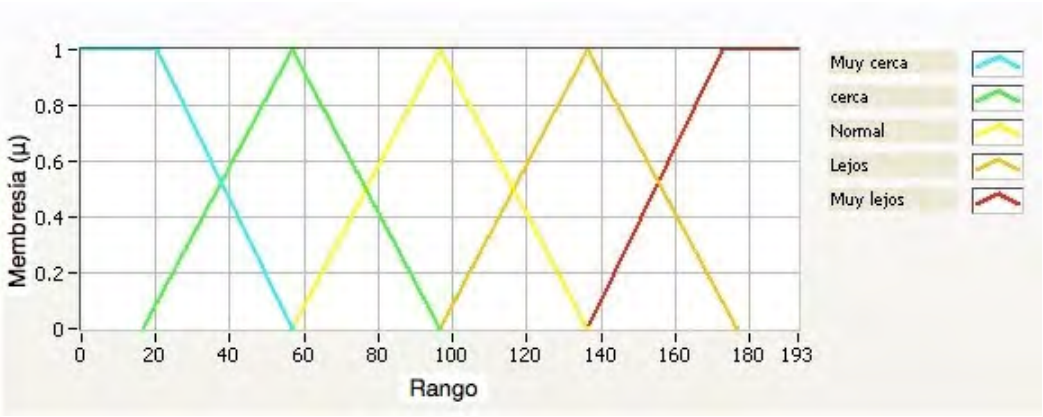

Figura 4.12 Variable de entrada “distanciaOrigen”

Al igual que la variable “distanciaObjeto”, esta variable tiene cinco funciones de membresía asociadas: “Muy cerca” que es una función trapezoidal, “Cerca”, “Normal”, “Lejos”, que son funciones triangulares y “Muy lejos”, función trapezoidal (Figura 4.12). El rango de operación está basado en el mismo concepto desarrollado anteriormente con la variable “*distanciaObjeto*”. Esta variable de entrada, se utiliza para el regreso del robot móvil al punto de referencia (origen).

• *anguloDeGiroOrigen*

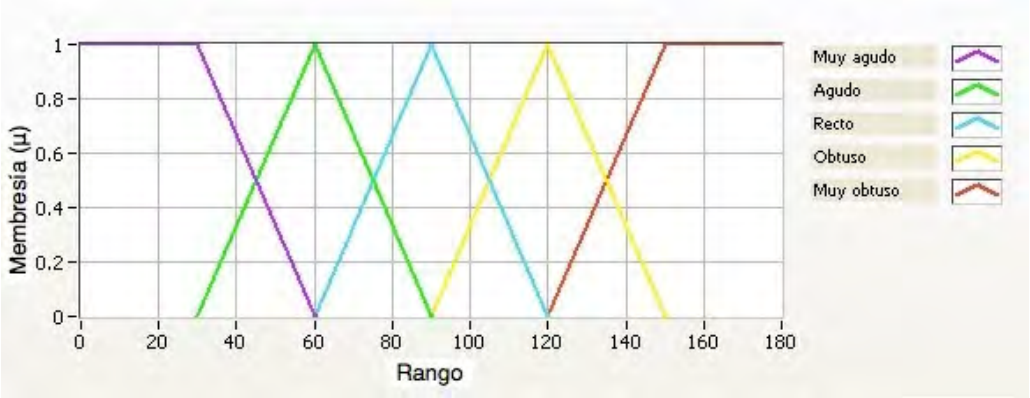

Figura 4.13 Variable de entrada “anguloDeGiroOrigen”

Esta variable cuenta con cinco funciones de membresía asociadas: “Muy agudo”, que es una función trapezoidal, “Agudo”, “Recto”, “Obtuso”, que son funciones triangulares y “Muy obtuso” que es una función trapezoidal (Figura 4.13). El rango de operación de esta variable es de 0 a 180 [º]. Y de igual manera que la variable “*anguloDeGiroObjeto*”, este rango se ha seleccionado en base al algoritmo capaz de decidir hacia qué dirección debe girar el robot móvil.

Variables de salida

Teniendo definidos los movimientos de desplazamiento que realizará el robot móvil, es posible determinar qué variables de salida son necesarias para el control del desplazamiento de éste. Dicho control se centra en los servomotores, es decir, la dirección en la que giran y la velocidad de giro. Existe una variable de salida más, la cuál permite controlar el rango de error en el cálculo del alineamiento del robot móvil con el objeto y con el origen; mientras más cerca esté del objeto o del origen, más grande será el valor de la variable de salida. Esto se debe a que en el proceso de adquisición de imágenes y en el proceso de cálculo de distancias y grados, existe un retraso que afecta al movimiento del robot móvil, lo que provoca menor precisión al estar más cerca del objeto.

En base a este análisis, se han determinado 6 variables de salida: "*velocidadMotoresObjeto*", "*giroMotoresObjeto*", "*errorAnguloObjeto*", "*velocidadMotoresOrigen*", "*giroMotoresOrigen*", "*errorAnguloOrigen*". Como en el caso de las variables de entrada, las variables de salida tienen también funciones de membresía asociadas y tienen un rango de operación. A continuación de describirá cada una de ellas:

• *velocidadMotoresObjeto*

Figura 4.14 Variable de salida "velocidadMotoresObjeto"

Esta variable de salida tiene cinco funciones de membresía asociadas: "Muy lento", que es una función trapezoidal, "Lento", "Normal", "Rápido", que son funciones triangulares y "Muy rápido" que es una función trapezoidal (Figura 4.14).

El rango de operación de esta variable fue determinado por las condiciones de operación en la adquisición de imágenes. El rango de valores que puede recibir el servomotor va de 0 a 100 y representa el porcentaje de la potencia con la que éste puede actuar, es decir, 100 representa el 100% de la potencia, 50 representa la mitad de la potencia, etc.

El valor máximo que puede tomar la variable de salida es 50 debido a que si se trabaja con la capacidad total del servomotor, el robot móvil se moverá con tal velocidad que el dispositivo de adquisición de imágenes no podrá percibir los cambios de movimiento de forma continua

y secuencial. Trabajando con la mitad de la potencia como valor máximo, la adquisición de imágenes resulta adecuada.

• *giroMotoresObjeto*

Figura 4.15 Variable de salida “giroMotoresObjeto”

Esta variable de salida tiene cinco funciones de membresía asociadas: “Giro muy lento”, que es una función trapezoidal, “Giro lento”, “Giro normal”, “Giro rápido”, que son funciones triangulares y “Giro muy rápido” que es una función trapezoidal (Figura 4.15). El rango de operación de esta variable está basado en lo explicado en la variable anterior (retraso en la adquisición de imágenes), con la diferencia de que el valor mínimo que puede tomar la variable de salida es 13; este valor es el que necesitan recibir los servomotores que gobiernan el sistema de locomoción para romper el par del robot móvil cuando está en reposo.

• *errorAnguloObjeto*

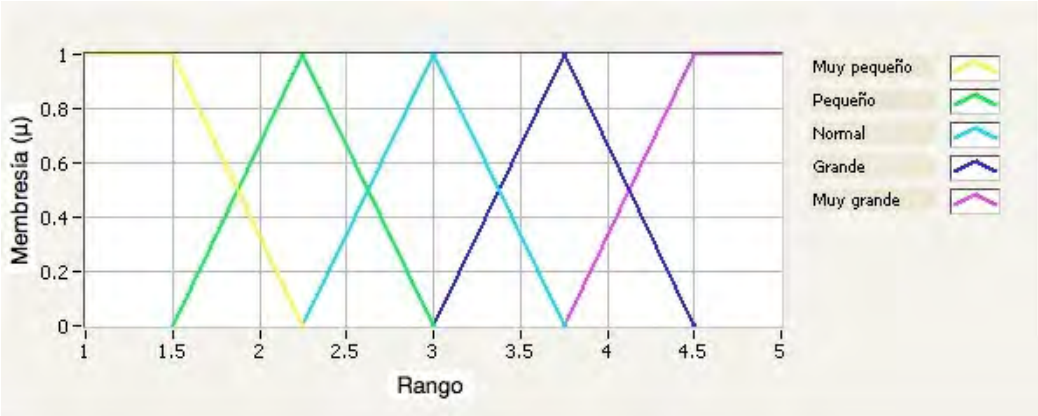

Figura 4.16 Variable de salida “errorAnguloObjeto”

Esta variable de salida tiene cinco funciones de membresía asociadas: “Muy pequeño”, que es una función trapezoidal, “Pequeño”, “Normal”, “Grande”, que son funciones triangulares y “Muy grande” que es una función trapezoidal (Figura 4.16). El rango de esta variable va de 1º

a 5º. El objetivo de esta variable es reducir el tiempo de alineamiento del robot móvil conforme se va acercando al objeto.

Debido al retraso en la adquisición de las imágenes es necesario compensar la variable de salida "*giroMotoresObjeto*", ya que el error de alineación es menor mientras más lejos esté el objeto, y aumenta al acercarse al mismo. Se ha elegido 5 como valor máximo que puede tomar esta variable, debido a que el alcance de sujeción de la garra, permite esta desviación.

• *velocidadMotoresOrigen*

Figura 4.17 Variable de salida "velocidadMotoresOrigen"

Esta variable de salida tiene cinco funciones de membresía asociadas: "Muy lento", que es una función trapezoidal, "Lento", "Normal", "Rápido", que son funciones triangulares y "Muy rápido" que es una función trapezoidal (Figura 4.17). El rango de operación de esta variable fue determinado por el mismo concepto desarrollado en la variable de salida análoga "*velocidadMotoresObjeto*".

• *giroMotoresOrigen*

Figura 4.18 Variable de salida "giroMotoresOrigen"

Esta variable de salida tiene cinco funciones de membresía asociadas: “Giro muy lento”, que es una función trapezoidal, “Giro lento”, “Giro normal”, “Giro rápido”, que son funciones triangulares y “Giro muy rápido” que es una función trapezoidal (Figura 4.18). El rango de operación de esta variable está basado en lo tratado en la variable análoga “*giroMotoresObjeto*”, salvo la diferencia del valor mínimo que puede tomar; en esta variable es 15 debido a que se considera que debe romper el par incluyendo el peso del objeto.

• *errorAnguloOrigen*

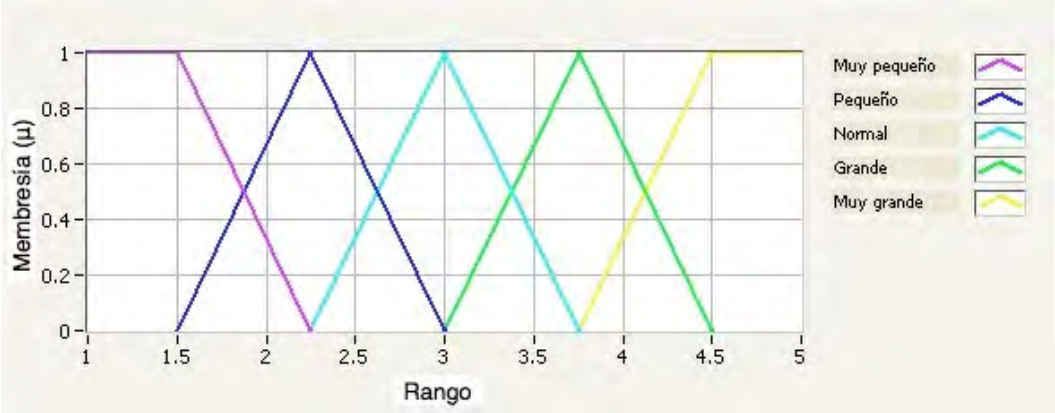

Figura 4.19 Variable de salida “errorAnguloOrigen”

Esta variable de salida tiene cinco funciones de membresía asociadas: “Muy pequeño”, que es una función trapezoidal, “Pequeño”, “Normal”, “Grande”, que son funciones triangulares y “Muy grande” que es una función trapezoidal (Figura 4.19). El rango de esta variable va de 1º a 5º y está determinado por todo lo especificado en la variable análoga “*errorAnguloObjeto*” con la diferencia de que está vinculado con el origen.

Base de reglas o de conocimiento

Como se ha visto anteriormente, la base de reglas o base de conocimiento junto con la máquina de inferencia, relaciona los conjuntos difusos de entrada con los conjuntos difusos de salida, es decir, define el comportamiento que gobierna al sistema. En base a las cuatro variables de entrada, a las seis variables de salida y a la respuesta de operación del robot móvil, se ha construido una base de reglas o de conocimiento que consta de 30 elementos. A continuación se enumeran las reglas, agrupadas de acuerdo a la variable de entrada y su relación con las variables de salida correspondientes.

• *distanciaObjeto*

1.- *IF ‘distanciaObjeto’ IS ‘Muy cerca’ THEN ‘velocidadMotoresObjeto’ IS ‘Muy lento’*
2.- *IF ‘distanciaObjeto’ IS ‘Cerca’ THEN ‘velocidadMotoresObjeto’ IS ‘Lento’*
3.- *IF ‘distanciaObjeto’ IS ‘Normal’ THEN ‘velocidadMotoresObjeto’ IS ‘Normal’*
4.- *IF ‘distanciaObjeto’ IS ‘Lejos’ THEN ‘velocidadMotoresObjeto’ IS ‘Rápido’*
5.- *IF ‘distanciaObjeto’ IS ‘Muy lejos’ THEN ‘velocidadMotoresObjeto’ IS ‘Muy rápido’*

6.- *IF ‘distanciaObjeto’ IS ‘Muy cerca’ THEN ‘errorAnguloObjeto’ IS ‘Muy grande’*
7.- *IF ‘distanciaObjeto’ IS ‘Cerca’ THEN ‘errorAnguloObjeto’ IS ‘Grande’*
8.- *IF ‘distanciaObjeto’ IS ‘Normal’ THEN ‘errorAnguloObjeto’ IS ‘Normal’*
9.- *IF ‘distanciaObjeto’ IS ‘Lejos’ THEN ‘errorAnguloObjeto’ IS ‘Pequeño’*
10.- *IF ‘distanciaObjeto’ IS ‘Muy lejos’ THEN ‘errorAnguloObjeto’ IS ‘Muy pequeño’*

• *anguloDeGiroObjeto*

11.- *IF ‘anguloDeGiroObjeto’ IS ‘Muy agudo’ THEN ‘giroMotoresObjeto’ IS ‘Giro muy lento’*
12.- *IF ‘anguloDeGiroObjeto’ IS ‘Agudo’ THEN ‘giroMotoresObjeto’ IS ‘Giro Lento’*
13.- *IF ‘anguloDeGiroObjeto’ IS ‘Recto’ THEN ‘giroMotoresObjeto’ IS ‘Giro Normal’*
14.- *IF ‘anguloDeGiroObjeto’ IS ‘Obtuso’ THEN ‘giroMotoresObjeto’ IS ‘Giro rápido’*
15.- *IF ‘anguloDeGiroObjeto’ IS ‘Muy obtuso’ THEN ‘giroMotoresObjeto’ IS ‘Giro muy rápido’*

• *distanciaOrigen*

16.- *IF ‘distanciaOrigen’ IS ‘Muy cerca’ THEN ‘velocidadMotoresOrigen’ IS ‘Muy lento’*
17.- *IF ‘distanciaOrigen’ IS ‘Cerca’ THEN ‘velocidadMotoresOrigen’ IS ‘Muy lento’*
18.- *IF ‘distanciaOrigen’ IS ‘Normal’ THEN ‘velocidadMotoresOrigen’ IS ‘Normal’*
19.- *IF ‘distanciaOrigen’ IS ‘Lejos’ THEN ‘velocidadMotoresOrigen’ IS ‘Rápido’*
20.- *IF ‘distanciaOrigen’ IS ‘Muy lejos’ THEN ‘velocidadMotoresOrigen’ IS ‘Muy rápido’*
21.- *IF ‘distanciaOrigen’ IS ‘Muy cerca’ THEN ‘errorAnguloOrigen’ IS ‘Muy grande’*
22.- *IF ‘distanciaOrigen’ IS ‘Cerca’ THEN ‘errorAnguloOrigen’ IS ‘Grande’*
23.- *IF ‘distanciaOrigen’ IS ‘Normal’ THEN ‘errorAnguloOrigen’ IS ‘Normal’*
24.- *IF ‘distanciaOrigen’ IS ‘Lejos’ THEN ‘errorAnguloOrigen’ IS ‘Pequeño’*
25.- *IF ‘distanciaOrigen’ IS ‘Muy Lejos’ THEN ‘errorAnguloOrigen’ IS ‘Muy pequeño’*

• *anguloGiroOrigen*

26.- *IF ‘anguloDeGiroOrigen’ IS ‘Muy agudo’ THEN ‘giroMotoresOrigen’ IS ‘Giro muy lento’*
27.- *IF ‘anguloDeGiroOrigen’ IS ‘Agudo’ THEN ‘giroMotoresOrigen’ IS ‘Giro Lento’*
28.- *IF ‘anguloDeGiroOrigen’ IS ‘Recto’ THEN ‘giroMotoresOrigen’ IS ‘Giro normal’*
29.- *IF ‘anguloDeGiroOrigen’ IS ‘Obtuso’ THEN ‘giroMotoresOrigen’ IS ‘Giro rápido’*
30.- *IF ‘anguloDeGiroOrigen’ IS ‘Muy obtuso’ THEN ‘giroMotoresOrigen’ IS ‘Giro muy rápido’*

Las variables de entrada, de salida y la base de reglas o conocimientos son guardadas en un archivo que tiene extensión “.fs”, este archivo debe vincularse con el VI que representa al controlodar difuso en la implementación de software.

Cabe mencionar que el método de desdifusión empleado en este controlador difuso fue el *Método del centroide* debido a la sencillez computacional que implica y a que arroja un resultado único.

4.4 Implementación.

Teniendo como base los elementos definidos en las secciones anteriores que constituyen el sistema difuso que gobierna el comportamiento del robot móvil, se ha desarrollado un programa en el entorno de programación LabVIEW 2010 Service Pack 1, versión 10.0.1f2, sobre el sistema operativo Microsoft Windows XP Profesional Service Pack 3.

Esta implementación de software está dividida en tres etapas:

- Etapa de adquisición de imágenes

- Etapa de procesamiento de datos

- Etapa de control

Las etapas están vinculadas de forma secuencial, es decir, la salida de una, es la entrada de la siguiente y así sucesivamente. La mayor parte de los procesos de cálculo están contenidos dentro de un ciclo *while*, sólo el proceso de abrir y configurar la comunicación con el dispositivo de adquisición de imágenes, el proceso de cerrar la comunicación y liberar la memoria, están fuera de ese ciclo porque sólo se necesitan ejecutar una vez.

A continuación se presenta el programa completo (Figura 4.20) y las etapas que lo componen, posteriormente se explicará cada una de ellas con más detalle:

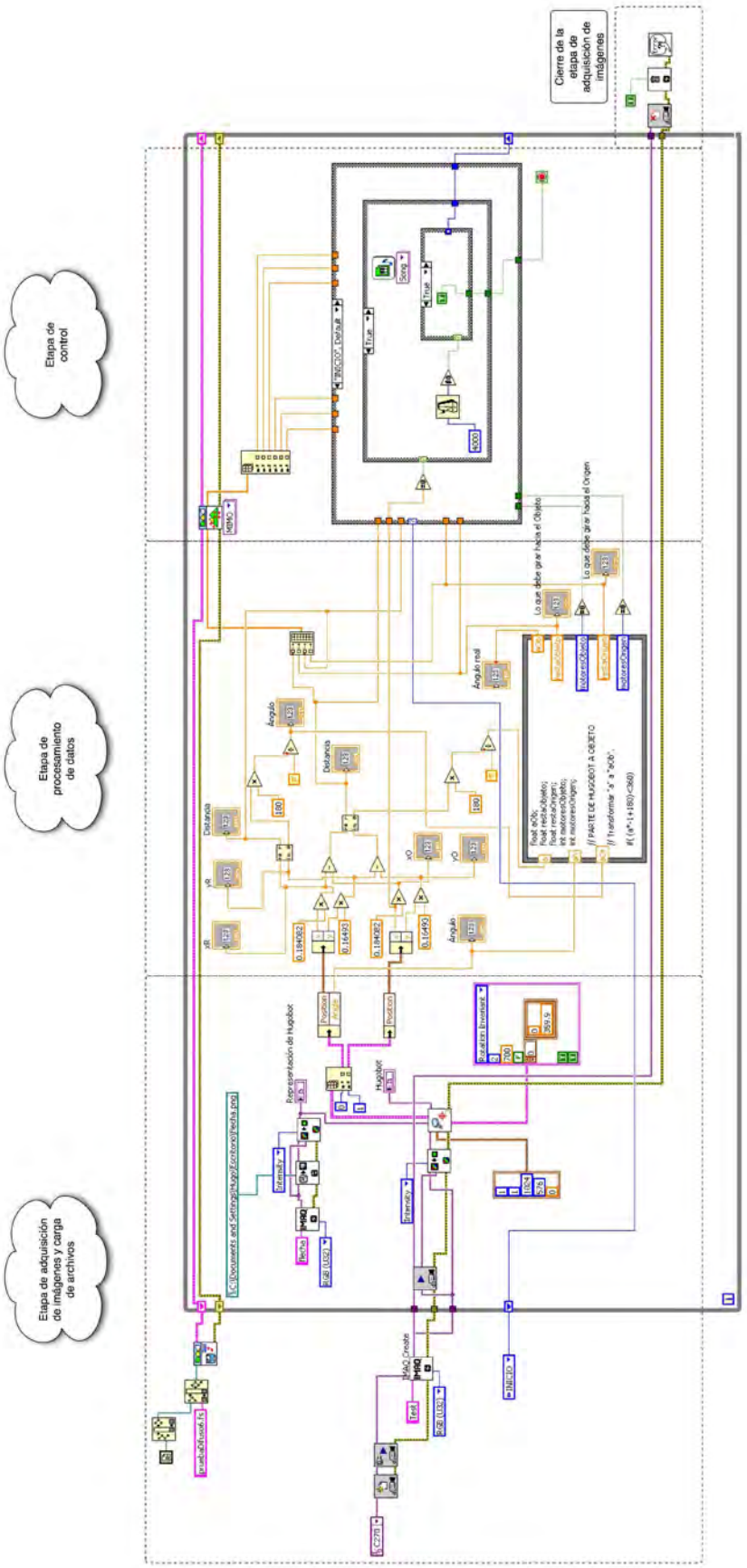

Figura 4.20 Programa completo que gobierna el comportamiento del robot móvil

•Etapa de adquisición de imágenes y carga de archivos

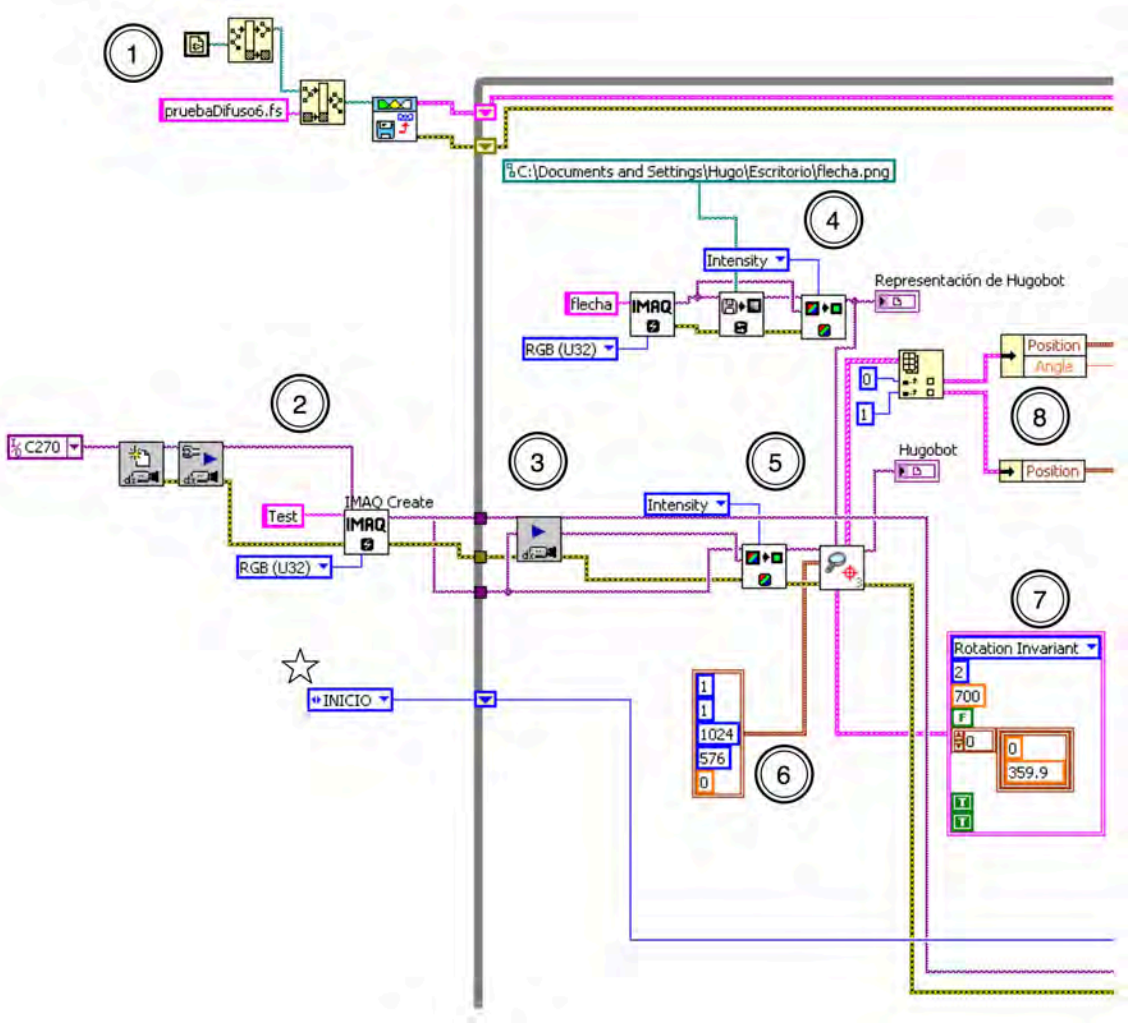

Figura 4.21a Etapa de adquisición de imágenes y carga de archivos

Como su nombre lo indica, en esta etapa se adquieren las imágenes y datos resultantes del reconocimiento de patrones que serán procesados en la siguiente etapa. Ahora se explica con más detenimiento cada sección de la Figura 4.21a:

Sección 1.- En esta sección se carga el archivo “.fs” en donde están almacenadas las variables de entrada, salida y la base de conocimiento con el cuál se construye el controlador difuso.

Sección 2.- Se abre la comunicación con la cámara, se configura, se inicializa la adquisición de imágenes y se crea un espacio en la memoria para guardar las imágenes que se adquieren con un cierto formato (*RGB (U32)*), es decir, se crea un buffer. Las imágenes adquiridas corresponden al área de acción del robot móvil.

Sección 3.- Adquiere el frame o imagen adquirida más actual desde el buffer anteriormente creado.

Sección 4.- Esta sección crea otro espacio en la memoria para almacenar un archivo imagen (patrón a buscar) cargado desde una ruta específica (*c:\Documents and Settings\Hugo \Escritorio\flecha.png*) extrayendo un plano simple de la imagen a color cargada. Usando el VI *IMAQ ExtractSingleColorPlane* y la opción *Intensity,* se extrae el plano de la intensidad de color, creando otra imagen con formato de 8 bits. Esta imagen es el patrón que se buscará en el área de acción del robot móvil (Figura 4.21b).

Figura 4.21b Forma usado para el reconocimiento de patrones

Sección 5.- De la imagen obtenida en la sección 3, se hace un procesamiento idéntico al de la imagen de la sección anterior, es decir, se extrae el plano de intensidad de color en una imagen con formato de 8 bits. En esta imagen se buscará el patrón configurado con anterioridad y se arrojarán los resultados de la búsqueda en clusters. Un cluster es un tipo de dato que está compuesto por varios elementos de distintos tipos; es el equivalente a una estructura de datos.

Sección 6.- Especifica las dimensiones en pixeles del área de búsqueda rectangular en la imagen. Las dimensiones seleccionadas son:

- Eje x: desde el pixel 1 al pixel 1024
- Eje y: desde el pixel 1 al pixel 576

En otras palabras, el área de búsqueda es de 1024 x 576 pixeles.

Sección 7.- Es un cluster en el que se configuran los parámetros de búsqueda de patrones. Estos parámetros son:

- *Rotation Invariant*: Busca el patrón en la imagen sin restricción de rotación del mismo.
- Number of Matches Requested: *Es el número de búsquedas válidas esperadas. Se han elegido 2 búsquedas válidas debido a que se desea encontrar el patrón que representa al robot y al correspondiente del objeto más cercano. El algoritmo de búsqueda opera con dos reglas básicas: a) buscae y encuentra el patrón más parecido al patrón referencia, b) a partir de éste, busca el siguiente patrón más parecido y más cercano al primer patrón encontrado, y así sucesivamente hasta no encontrar más patrones. De esta forma, se resuelve el problema de elección de cuál objeto se debe recoger en caso de ser varios.*
- *Minimum Match Score*: Es el valor mínimo que debe tener un patrón para considerarse válido. 0 equivale a no parecerse, 1000 equivale a ser idéntico. Se ha elegido el valor 700 para esta configuración.

• *Subpixel Accuracy*: Determina si el resultado de la búsqueda del patrón se devuelve con precisión de subpixeles. Este parámetro no está activo debido a que no es necesaria esa precisión.

• *Rotation Angle Ranges*: Se especifica el rango de grados de rotación que se espera encontrar en el patrón a buscar. Este rango está especificado con el ángulo inicial (*0º*) y con el ángulo final (*359.9º*). Este rango fue seleccionado para no provocar conflictos en el cálculo de grados ya que de forma práctica 0º = 360ª.

• *Show Search area*: Especifica si el centro y el rectángulo que delimita al patrón encontrado está en el área de búsqueda. Este parámetro está activo debido a que ayuda a la localización del patrón.

• *Show Result*: Muestra la localización del centro y el rectángulo que delimita al patrón encontrado sobre el área de búsqueda. Este parámetro está activo debido a que muestra visualmente la localización del patrón.

Sección 8.- Los resultados de la búsqueda de patrones vienen en un conjunto de clusters que se acomodan en un arreglo unidimensional para extraer de forma individual cada cluster con la información de los resultados de búsqueda. Al primer cluster (correspondiente al robot móvil), se le extrae la siguiente información: posición (x,y) y ángulo. Al segundo cluster (correspondiente al objeto más cercano), sólo se le extrae la posición (x,y).

En la parte media izquierda de la Figura 4.21a está presente el símbolo ☆ el cuál indica que esa sección pertenece a otra parte del programa (etapa de control), más adelante se hablará de ella.

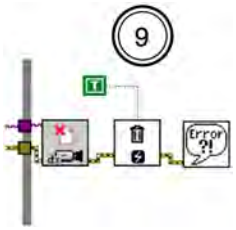

Figura 4.21c Sección final de la etapa de adquisición de imágenes

Sección 9.- Es la parte final de esta etapa, en ella se cierra la comunicación con el dispositivo de adquisición, detiene la adquisición de imágenes y libera los recursos asociados con la adquisición. Se libera también el espacio que antes fue reservado para las imágenes adquiridas, es decir, el buffer y finalmente muestra un mensaje de error en todo el proceso antes descrito si es que existe. Esta sección está fuera del ciclo *while* (Figura 4.21c).

•Etapa de procesamiento de datos

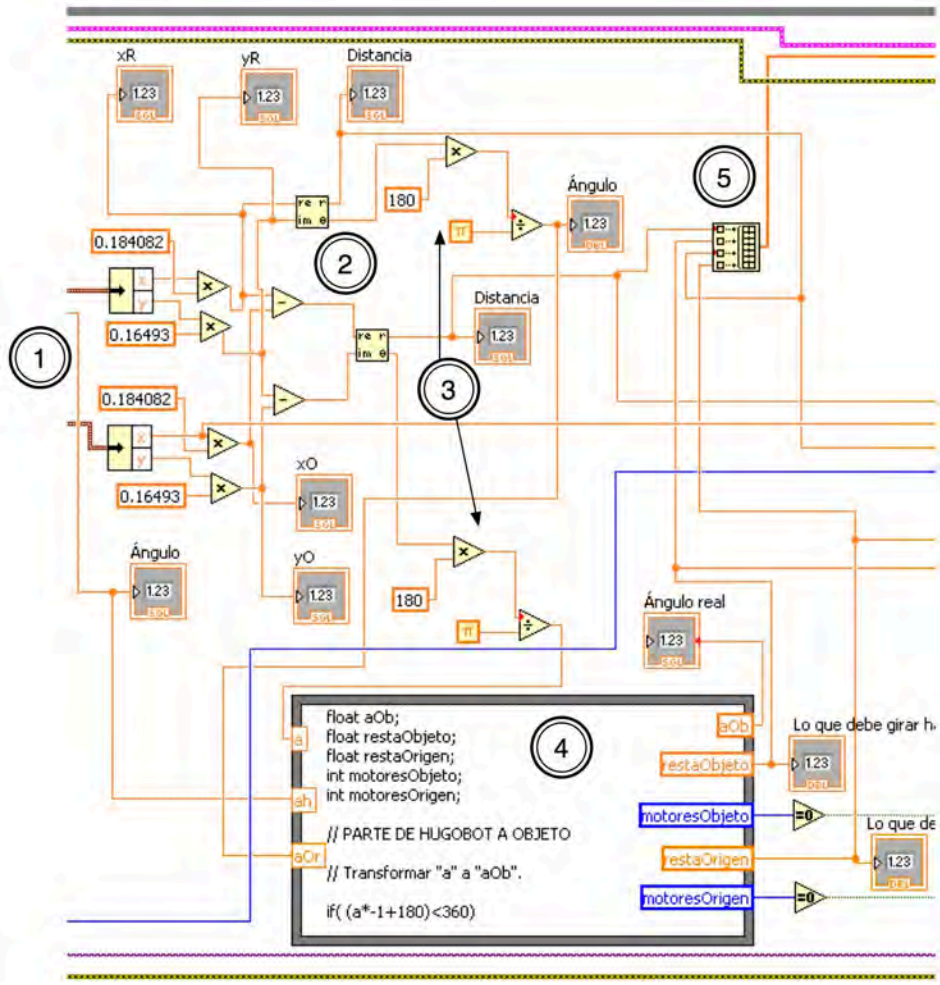

Figura 4.22 Etapa de procesamiento de datos

En esta etapa se reciben los datos recolectados en la etapa de adquisición; se procesan y acondicionan para ser usados posteriormente por el controlador difuso. A continuación, se explica cada sección con más detalle (Figura 4.22).

Sección 1.- Se hace una conversión de pixeles a centímetros con un factor calculado de la siguiente forma:

Para el eje x:

$$1024 \text{ pixeles} \rightarrow 165 \text{ [cm]}$$
$$1 \text{ pixel} \rightarrow x \text{ [cm]}$$

Resolviendo la regla de 3:

$$x = \frac{(1)(165)}{1024}$$
$$x = 0.161132$$

Por lo tanto:

1 pixel = 0.161132 [cm]

Para el eje y:

576 pixeles → 94 [cm]
1 pixel → ? [cm]

Resolviendo la regla de 3:

$$x = \frac{(1)(94)}{576}$$
$$x = 0.163194$$

Por lo tanto:

1 pixel = 0.163194x [cm]

Con este factor, es posible calcular la distancia en centímetros que existe entre el robot móvil y el objeto a recolectar. Estos cálculos están realizados sobre el sistema cartesiano de coordenadas, es decir con las componentes (x,y), y se aplica el mismo procedimiento para las coordenadas del robot móvil y las coordenadas del objeto.

Sección 2.- En esta sección, se hace una conversión geométrica de coordenadas cartesianas (*x,y*) a coordenadas polares (*r,θ*). La forma de hacer dicha conversión es usando las siguientes ecuaciones:

$$r = \sqrt{x^2 + y^2}$$
$$\theta = ang\tan(\frac{y}{x})$$

En la parte superior de esta sección, se hace la conversión de las coordenadas del robot móvil con respecto al origen. En la parte inferior de la sección, se hace otra conversión de coordenadas del objeto a recolectar con respecto al robot móvil. Esto se logra restando la posición del robot móvil menos la posición del objeto, es decir:

$$r_{or} = \sqrt{(x_{robot} - x_{objeto})^2 + (y_{robot} - y_{objeto})^2}$$

$$\theta_{or} = ang\tan(\frac{y_{robot} - y_{objeto}}{x_{robot} - x_{objeto}})$$

Cabe mencionar que las unidades resultantes de la componente *θ* (ángulo) son radianes.

Sección 3.- La componente *θ* de las posiciones del robot móvil y del objeto de la sección anterior son convertidas a grados, esto se lleva a cabo con la siguiente operación:

$$x = \frac{180(p)}{\pi}$$

Donde "*p*" es el valor de la componente *θ* en radianes y "*x*" es el valor en grados [º].

Sección 4.- En esta parte está contenida en un nodo de fórmula, en el cual se pueden evaluar fórmulas matemáticas y expresiones similares a las del lenguaje de programación C. El código es un algoritmo que permite al robot móvil decidir en qué sentido girará para alinearse con el objeto de forma más rápida y con el menor ángulo de rotación posible. A continuación se presenta el código:

```
// Declaración de variables auxiliares

float aOb;
float restaObjeto;
float restaOrigen;
int motoresObjeto;
int motoresOrigen;


// PARTE DEL ROBOT MÓVIL AL OBJETO

// Transformar "a" a "aOb".  Se hace esta operación debido a que "a" es un ángulo que está
// dado en sentido horario y es negativo.

if( (a*-1+180)<360)

   aOb=(a*-1)+180;

else

   aOb=(a*-1+180)-360;
```

```
// Grados y dirección que debe girar con respecto al objeto.

restaObjeto=aOb-ah;

if(restaObjeto>0 && restaObjeto<=180){

    motoresObjeto=0;              // El “0” representa giro a la izquierda.
}

else{

     if(restaObjeto>180 && restaObjeto<=360){

       restaObjeto=restaObjeto*-1+360;
       motoresObjeto=1;        // El “1” representa giro a la derecha.
     }

     else{

      restaObjeto=restaObjeto*-1;
      motoresObjeto=1;         // El “1” representa giro a la derecha.
      }
}


// PARTE DE ROBOT MÓVIL AL ORIGEN

//Grados y dirección que debe girar con respecto al origen.

restaOrigen=180-aOr-ah;      // El resultado de esta resta está dado en sentido horario; si se
                             // suma 180, el resultado estará en sentido antihorario.

if (restaOrigen>0 && restaOrigen<=180){

    motoresOrigen=0; // El “0” representa giro a la izquierda
}

else{

     if(restaOrigen>180 && restaOrigen<=360){

       restaOrigen=restaOrigen*-1+360;
       motoresOrigen=1;        // El “1” representa giro a la izquierda.
     }

     else{

          if(restaOrigen<=-180) {

            restaOrigen=restaOrigen+360;
            motoresOrigen=0;   // El “0” representa giro a la izquierda.
           }

          else{
```

```
            restaOrigen=restaOrigen*-1;
            motoresOrigen=1;    // El “1” representa giro a la derecha.
          }

      }
}
```

Sección 5.- Los cuatro datos obtenidos de la posición del robot móvil y del objeto a recolectar se insertan en un arreglo que será una de las entradas de la etapa siguiente. Estos datos son los valores correspondientes a las cuatro variables de entrada del controlador difuso. En orden descendente, los valores de las variables están acomodados en el arreglo de la siguiente forma:

I.- valores de la variable “*distanciaObjeto”*
II.- valores de la variable “anguloDeGiroObjeto”
III.- valores de la variable “distanciaOrigen”
IV.- valores de la variable “anguloDeGiroOrigen”

•Etapa de control

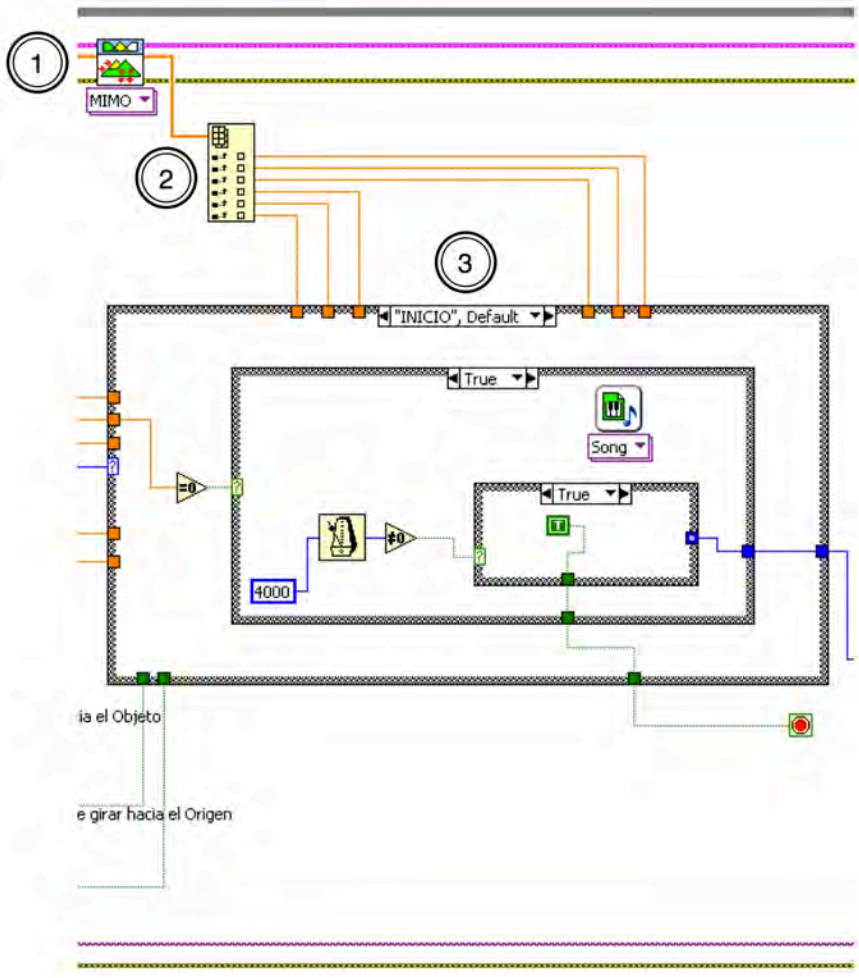

Figura 4.23 Etapa de control

Es la última etapa de la implementación, se reciben los datos de la etapa anterior que son las entradas del controlador difuso. Los valores que arroja el controlador gobiernan al sistema de locomoción y sujeción del robot móvil. Una máquina de estados define en qué momento son usados estos valores y qué actuador se acciona para llevar a cabo los movimientos necesarios que llevarán al robot móvil a lograr su objetivo. A continuación se explica con detalle, cada sección de esta fase (Figura 4.23):

Sección 1.- Este VI representa al controlador difuso. En la primer fase (Etapa de adquisición de imágenes y carga de archivos) se le asoció un archivo que contiene las variables de entrada, de salida y la base de reglas o conocimiento; en la segunda etapa (Etapa de procesamiento de datos) se creó un arreglo con datos que son los valores correspondientes a las cuatro variables de entrada que es la entrada principal del controlador difuso. La salida de este VI es un arreglo con datos que son los valores correspondientes a las seis variables de salida definidas anteriormente.

Sección 2.- Al arreglo que proviene del VI que representa al controlador difuso, se le extrae cada valor que contiene. En orden descendente, los valores de las variables están acomodados en el arreglo de la siguiente forma:

I.- valores de la variable "*velocidadMotoresObjeto"*
II.- valores de la variable *"giroMotoresObjeto"*
III.- valores de la variable *"errorAnguloObjeto"*
IV.- valores de la variable *"velocidadMotoresOrigen*"
V.- valores de la variable "*giroMotoresOrigen*"
VI.- valores de la variable "*errorAnguloOrigen*"

Estos valores serán usados por la máquina de estados.

Sección 3.- Es la máquina de estados que define la secuencia de operación del robot móvil. Recibe los valores de salida del controlador difuso para controlar los sistemas de locomoción y sujeción en determinados momentos y cumpliendo ciertas condiciones. La máquina de estados cuenta con diez estados (Figura 4.24):

a) *INICIO*
b) *AlineaObjeto*
c) *AvanzaObjeto*
d) *DetectaObjeto*
e) *RegresaUnPoco*
f) *AgarraObjeto*
g) *AlineaOrigen*
h) *AvanzaOrigen*
i) *SueltaObjeto*
j) *Fin*

A continuación, se muestra la representación de la máquina mencionada y se explica de forma más detallada cada estado:

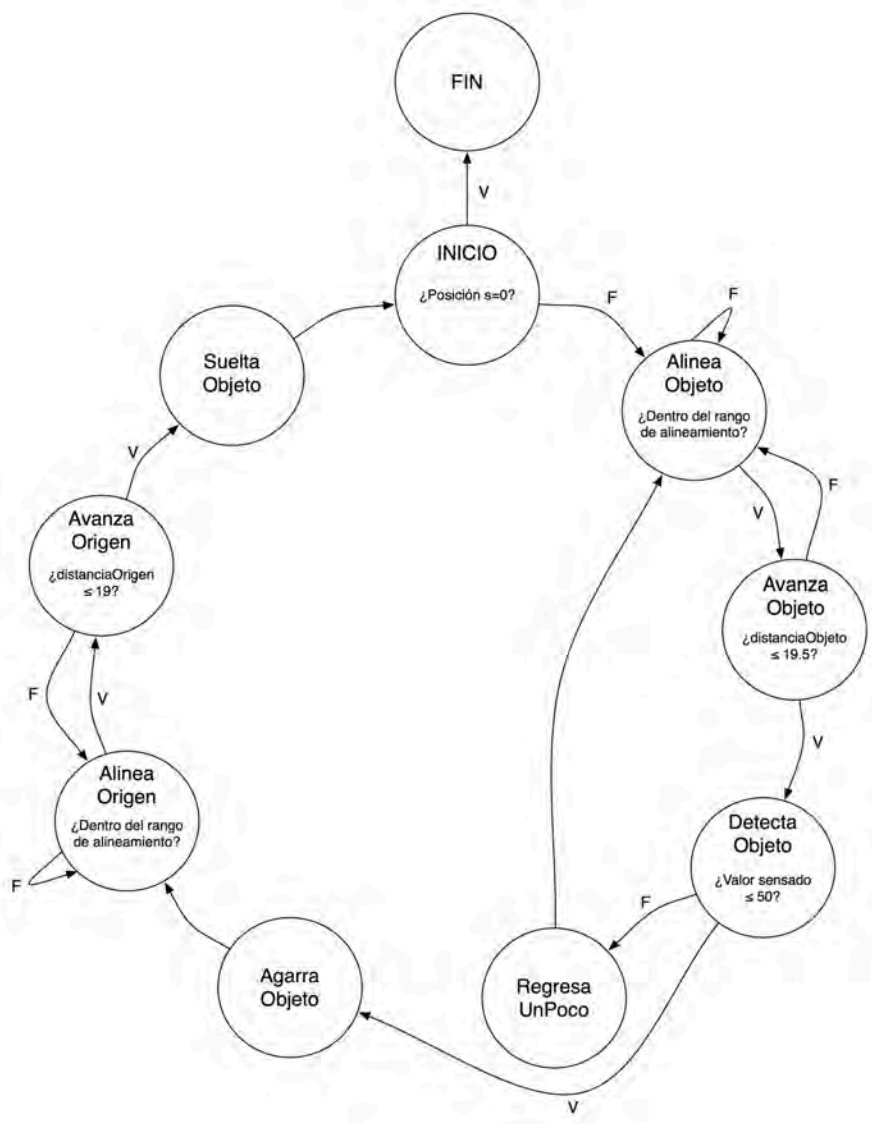

Figura 4.24 Máquina de estados que gobierna el comportamiento del robot móvil

- INICIO

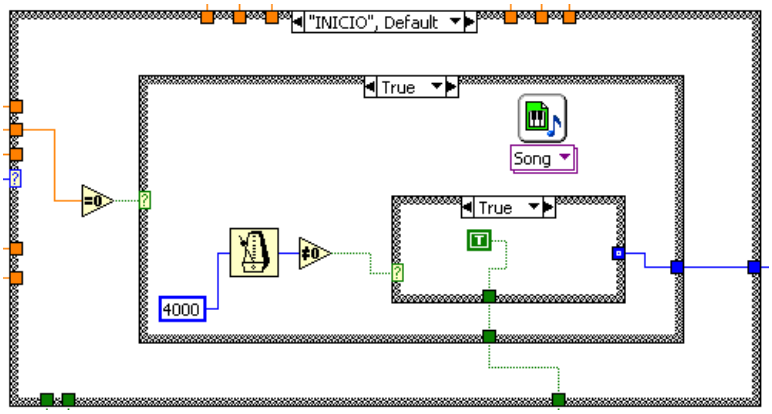

Figura 4.25 Estado INICIO

Es el estado inicial, en él se valida que al menos haya un objeto a recolectar. Esto se lleva a cabo comparando el valor de la componente “x” de la posición del objeto con el valor numérico cero, en caso verdadero la máquina de estados finaliza, es decir, va al estado FIN y reproduce un archivo de audio indicando que la ejecución ha finalizado, en caso contrario, avanza al siguiente estado (Figura 4.25).

- AlineaObjeto

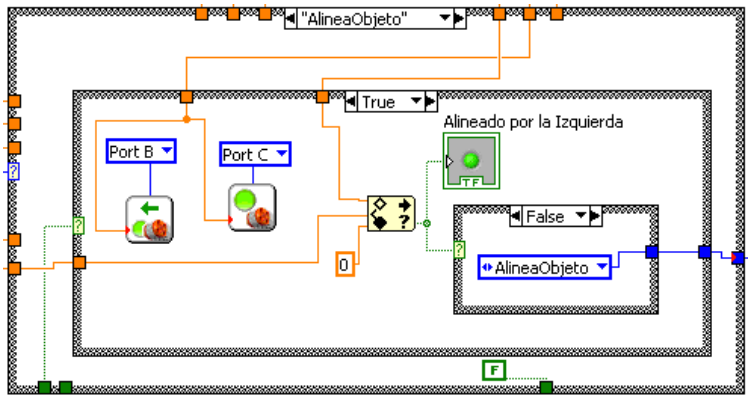

Figura 4.26a Estado AlineaObjeto (Izquierda)

Es el segundo estado, recibe los valores de las variables de salida “*giroMotoresObjeto*” y “*errorAnguloObjeto*”. Los valores de la primer variable controlan la rapidez de los motores que giran en sentido opuesto uno del otro, provocando la rotación del robot móvil. Los valores de la segunda variable controlan el rango de compensación de alineación del robot móvil. Tiene dos posibles formas de alinearse: por la izquierda (Figura 4.26a) o por la derecha (Figura 4.26b), esta decisión está gobernada por una salida del nodo de fórmula (motoresObjeto). Cuando la condición de alineación se cumple, avanza al siguiente estado, de lo contrario permanece en este estado.

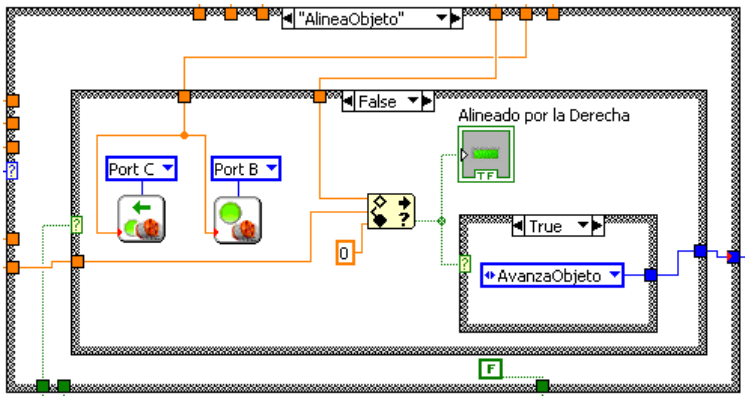

Figura 4.26b Estado AlineaObjeto (Derecha)

- AvanzaObjeto

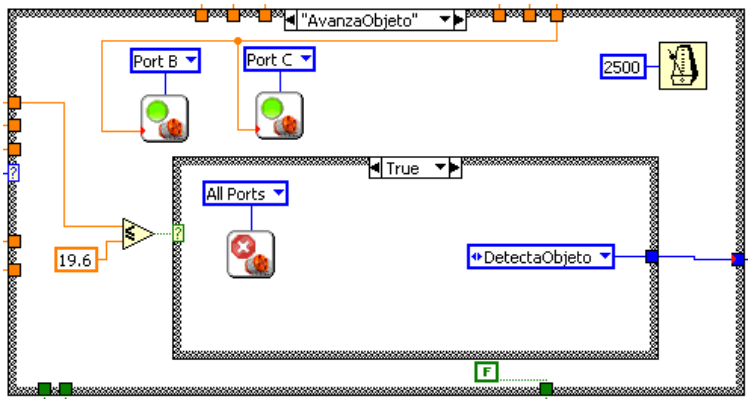

Figura 4.27 Estado AvanzaObjeto

Es el tercer estado, recibe los valores de la variable de salida “velocidadMotoresObjeto” que controlan la rapidez de los motores que giran en el mismo sentido para trasladarse de forma rectilínea en dirección al objeto. Este estado está activo por 2500 [ms], después de este tiempo, el valor de la distancia entre el robot móvil y el objeto se compara con 19.6 para saber si es menor o igual, en caso verdadero, avanza al siguiente estado, de lo contrario regresa al estado anterior (Figura 4.27).

- DetectaObjeto

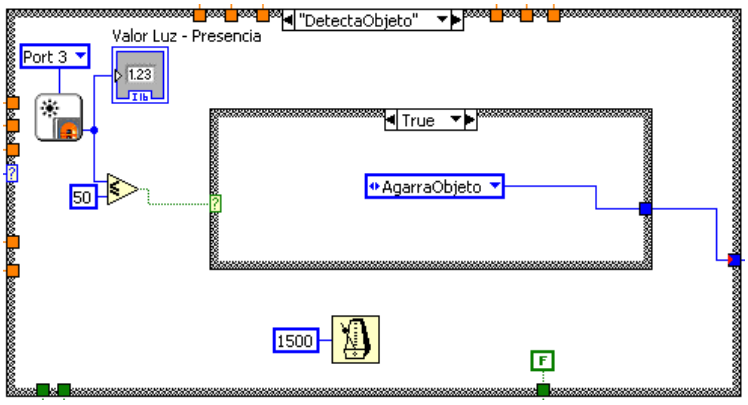

Figura 4.28 Estado DetectaObjeto

Es el cuarto estado, en él, se sensa la cantidad de luz y el valor que arroja el sensor es comparado con 50, si el valor es menor o igual, avanza al estado “AgarraObjeto”, de lo contrario avanza al estado “RegresaUnPoco”. Este estado es una verificación para determinar si el objeto está en el rango de sujeción de la garra. En caso de no estar en el rango, se repetirá el proceso de alineamiento y traslación (Figura 4.28). Este estado está activo por 1500 [ms].

- RegresaUnPoco

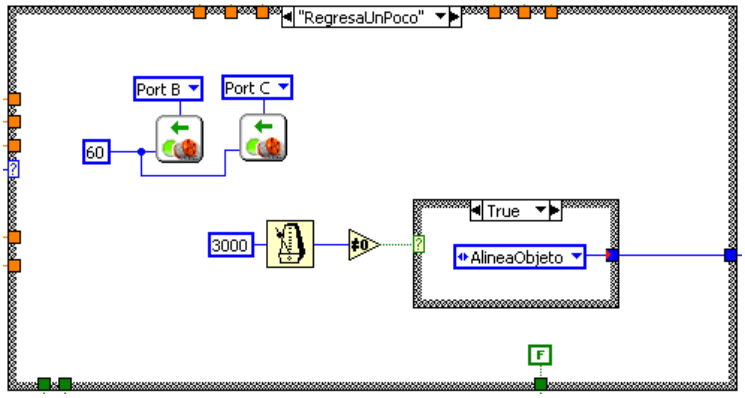

Figura 4.29 Estado RegresaUnPoco

Es el quinto estado; no tiene condicionante, el propósito es alejar al robot móvil del objeto para que realice nuevamente el proceso alineamiento y translación. Este estado está activo durante 3000 [ms] y en ese tiempo, los motores que rigen el sistema de locomoción giran en sentido inverso con una potencia del 60% (Figura 4.29). Al término del lapso, avanza al estado “AlineaObjeto”.

- AgarraObjeto

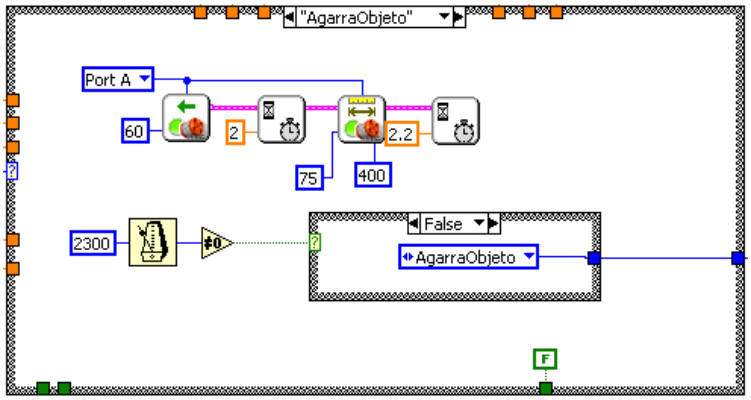

Figura 4.30 Estado AgarraObjeto

Es el sexto estado, el sistema de sujeción entra en acción con el objeto. El servomotor que gobierna este sistema es accionado de forma inversa con una potencia del 60% y espera 2 segundos para garantizar la máxima apertura de la garra, en seguida se acciona en sentido opuesto con una potencia del 75% girando 400 grados haciendo que la garra se cierre. Al llegar a este punto, el objeto ya está sujeto y continúa al siguiente estado (Figura 4.30).

- AlineaOrigen

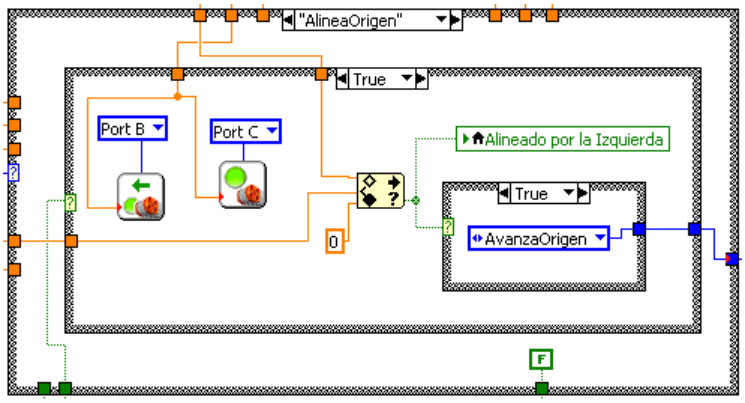

Figura 4.31a Estado AlineaOrigen (Izquierda)

Es el séptimo estado el cuál controla el regreso del robot móvil al origen. Recibe los valores de las variables de salida “giroMotoresOrigen” y “errorAnguloOrigen”. Al igual que en su variable análoga, los valores de la primer variable controlan la rapidez de los motores que giran en sentido opuesto uno del otro, provocando la rotación del robot móvil. Los valores de la segunda variable controlan el rango de compensación de alineación para considerar al robot móvil alineado. Tiene dos posibles formas de alinearse: por la izquierda (Figura 4.31a) o por la derecha (Figura 4.31b), esta decisión está gobernada por una salida del nodo de fórmula (motoresOrigen). Cuando la condición de alineamiento se cumple, avanza al siguiente estado, de lo contrario permanece en este estado.

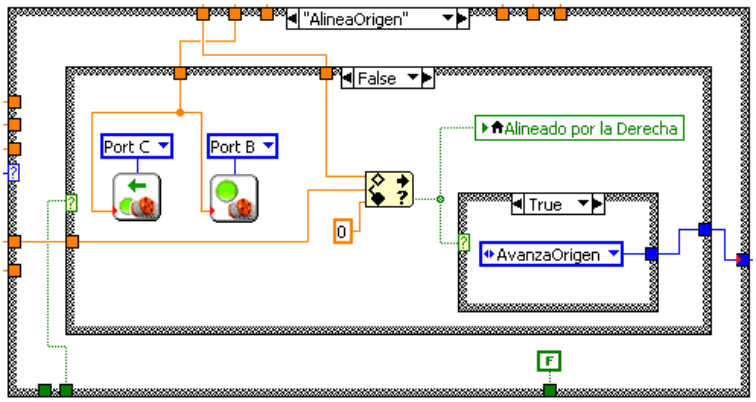

Figura 4.31b Estado AlineaOrigen (Derecha)

- AvanzaOrigen

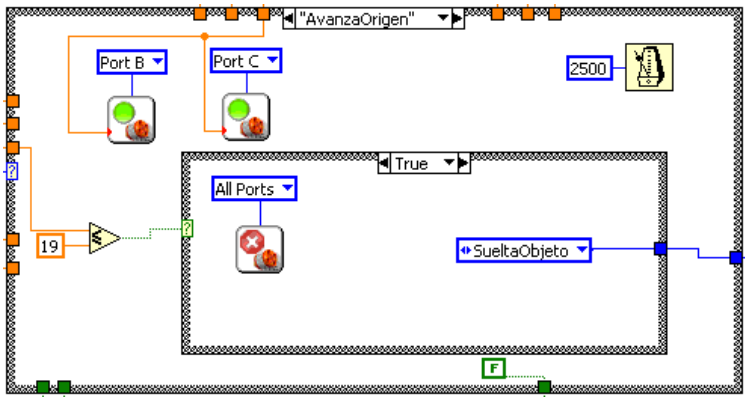

Figura 4.32 Estado AvanzaOrigen

Es el octavo estado, al igual que su estado análogo, recibe los valores de la variable de salida “velocidadMotoresOrigen” que controlan la rapidez de los motores que giran en el mismo sentido para trasladarse de forma rectilínea en dirección al origen. Este estado está activo por 2500 [ms], después de este tiempo, el valor de la distancia entre el robot móvil y el objeto se compara con 19 para saber si es menor o igual, en caso verdadero, avanza al siguiente estado, de lo contrario regresa al estado anterior (Figura 4.32).

- SueltaObjeto

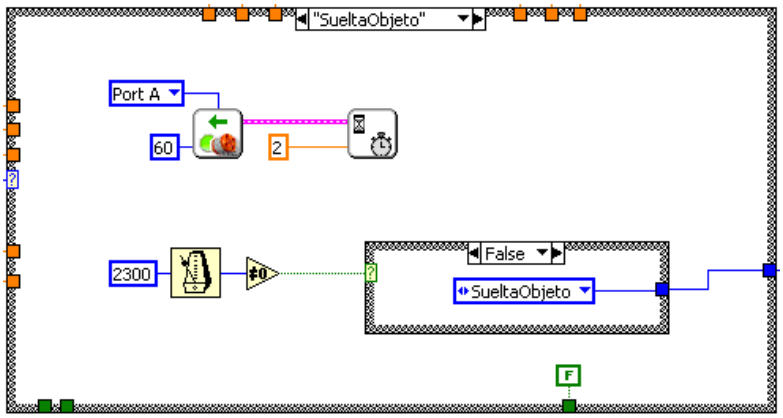

Figura 4.33 Estado SueltaObjeto

Es el último estado, el sistema de sujeción actúa una vez más liberando el objeto muy cerca del origen. Esto se lleva a cabo accionando el servomotor de forma inversa con una potencia del 60%, espera dos segundos y al cabo de este tiempo, avanza al estado INICIO (Figura 4.33).

Cuando la máquina de estados llega a este punto, se ha completado el ciclo de recolectar el objeto, en el caso que hubiese más objetos, la máquina de estados comienza una vez más y así continua hasta que recolecte el último objeto detectado.

La sección referida con el símbolo  en la “etapa de adquisición de imágenes y carga de archivos”, forma parte de la máquina de estados. Esta sección se encarga de inicializarla cuando todo el programa comienza, de esta forma se garantiza que el estado INICIO sea el primero en ejecutarse.

Capítulo 5
Resultados

5.1 Área de visión (datos experimentales).

Cómo se vio en el capítulo anterior, el rango de visión de la cámara es un rectángulo que tiene 165 [cm] de largo y 94 [cm] de ancho (Figura 5.1).

Figura 5.1 Rango de visión de la Cámara Web

Sin embargo, debido a las características de la lente del dispositivo de adquisición de datos (cámara web), a cierta distancia y a cierto ángulo de visión, los objetos y formas que vislumbra tienden a deformarse.

El objetivo (lente) es el encargado de definir la imagen en el sensor de un dispositivo de adquisición de imágenes. Existen muchos tipos de objetivos, algunos de los más usados son:

- Ojo de pez
- Gran angular
- Objetivo normal
- Teleobjetivo
- Objetivo macro

El tipo de objetivo que corresponde a la cámara web es el gran angular. Este tipo de objetivos tienen mucha profundidad de campo, poseen un ángulo de visión que va de los 63º a los 180º y producen distorsión en los bordes del encuadre debido a la curvatura de la lente.

Debido al efecto de distorsión descrito anteriormente, el cálculo de las distancias de los objetos que se encuentran situados cerca de los bordes del rango de visión resulta impreciso; mientras más cerca de un borde, la distancia del objeto en perspectiva con

respecto a la lente aumenta, mientras más cerca del centro, la distancia del objeto en perspectiva con respecto a la lente disminuye.

Se realizaron diversas pruebas de efectividad en la sujeción de objetos del robot móvil en distintas áreas del rango de visión de la cámara web (Figura 5.2). Para poder cuantificar estas pruebas, el rango de visión fue divido en 15 cuadrantes de 33 [cm] de largo y 31.33 [cm] de ancho:

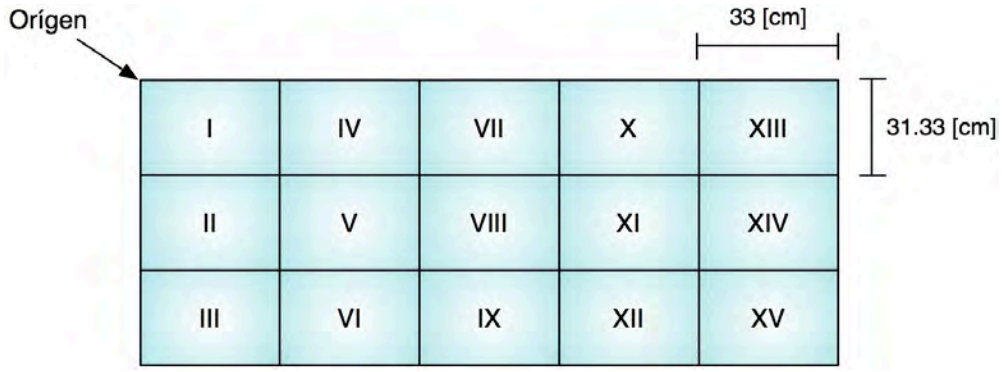

Figura 5.2 División del rango de visión en 15 cuadrantes

Por cada cuadrante, se realizaron 20 muestras. A continuación se muestra la tabla de frecuencias por cada cuadrante, seguido de un histograma del comportamiento del robot móvil que se ha observado.

Cuadrante	Aciertos de sujeción
I	Cuadrante base
II	7/20
III	6/20
IV	7/20
V	15/20
VI	14/20
VII	8/20
VIII	16/20
IX	15/20
X	7/20
XI	17/20

Cuadrante	Aciertos de sujeción
XII	14/20
XIII	5/20
XIV	15/20
XV	14/20

Tabla 5.1 Tabla de frecuencias de efectividad de sujeción del robot móvil por cuadrante

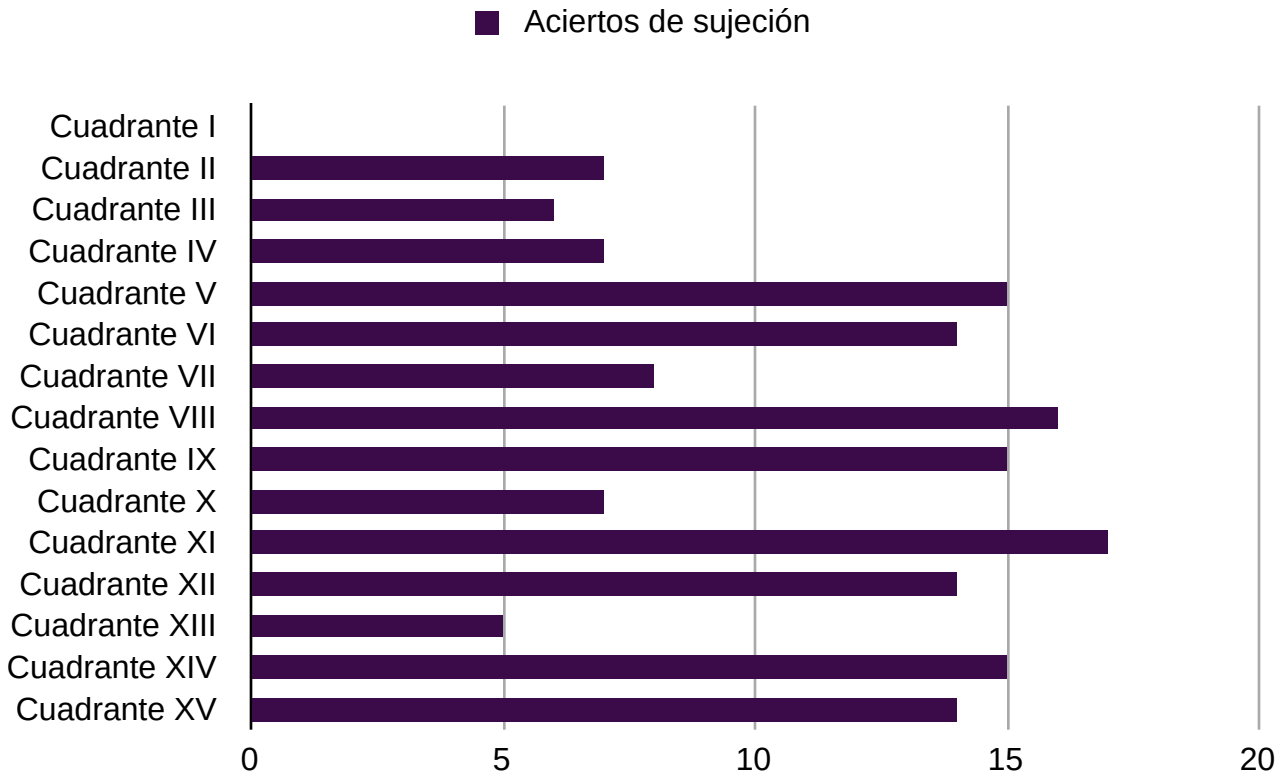

Figura 5.3 Histograma de los aciertos de sujeción por cuadrante

Con los resultados obtenidos en la toma de muestras, se determinaron los valores de distancia óptimos que complementan el cálculo de la distancia de los objetos y que hacen más confiable la sujeción de objetos del robot móvil por cuadrante. Estos valores se muestran en la siguiente tabla:

Cuadrante	Valor determinado [cm]
I	-
II	23.5

Cuadrante	Valor determinado [cm]
III	23.5
IV	23
V	23
VI	22
VII	19.5
VIII	19.5
IX	19.5
X	16
XI	16
XII	16
XIII	14
XIV	14
XI	14

Tabla 5.2 Tabla de valores que hace más confiable la sujeción del objeto por cuadrante

Es importante mencionar que al tomar las muestras por cada cuadrante, la posición del objeto era aleatoria, es decir, el objeto era situado en cualquier parte dentro del cuadrante. Cuando estas muestras se tomaron en los cuadrantes de los extremos, hubo áreas “muertas” en las que el patrón no era reconocido debido al efecto de distorsión mencionado con anterioridad, la siguiente figura muestra el área considerada de operación del robot móvil basada en la toma de muestras (Figura 5.4).

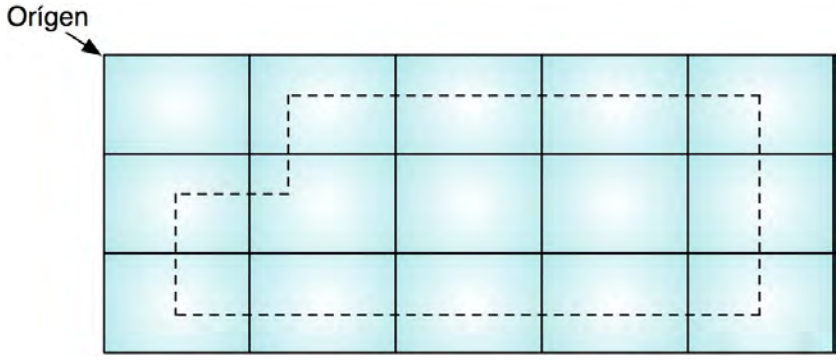

Figura 5.4 Área de operación del robot móvil

5.2 Ángulo de visión.

En la figura anterior se mostró el área de operación del robot móvil basada en la toma de muestras, el límite de dicha área está determinado por el ángulo de visión del dispositivo de adquisición de imágenes en el cual el patrón es detectado. Este ángulo de visión tiene dos componentes que si orientamos con respecto al sistema de referencia del área de visión, serían *ángulo de visión con respecto a "x"* y *ángulo de visión con respecto a "y"*.

El ángulo de visión fue determinado a partir de la distancia que existe con respecto del centro del área de visión y el punto en donde el patrón deja de reconocerse, la Figura 5.5 lo muestra de forma gráfica:

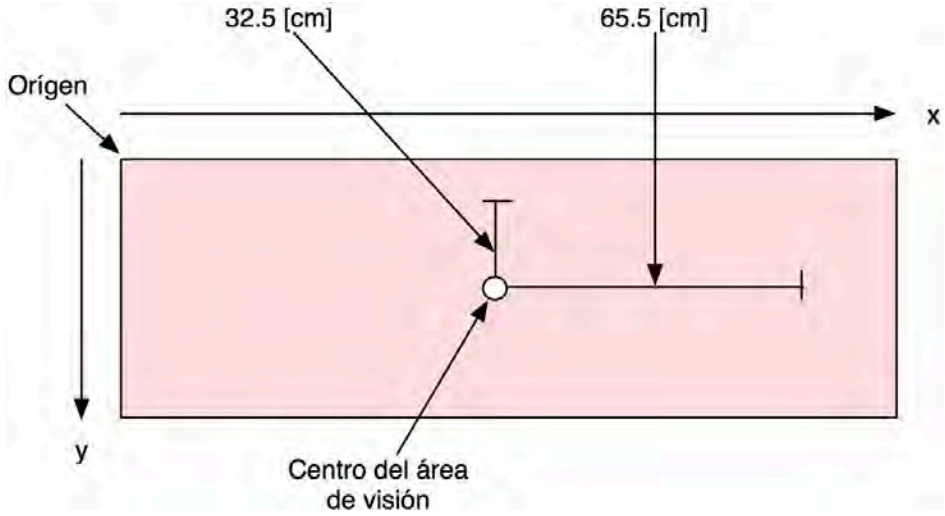

Figura 5.5 Centro del área de visión y medidas para calcular el ángulo de visión de la cámara web

Para el cálculo del *ángulo de visión con respecto al eje "x"*, se plantea un triángulo rectángulo con las distancias que van del centro del área de visión hacia la cámara web (cateto adyacente) y del centro del área de visión al límite del área de operación del robot móvil con respecto al eje "x" (cateto opuesto).

A continuación, se utiliza la identidad trigonométrica *tangente* y se obtiene el *ángulo de visión con respecto al eje "x"* (Figura 5.6):

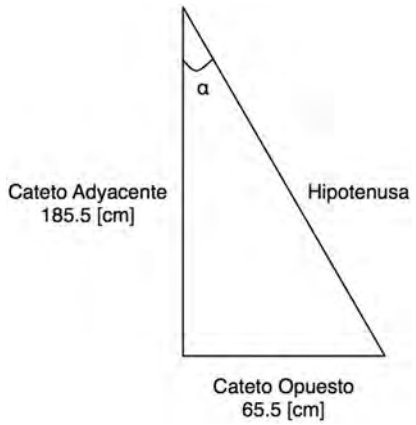

Figura 5.6 Triángulo formado para calcular el ángulo de visión de la cámara con respecto al eje x

$$\tan\alpha = \frac{c.o.}{c.a.}$$

$$\alpha = \tan^{-1}\left(\frac{c.o.}{c.a.}\right)$$

$$\alpha = \tan^{-1}\left(\frac{65.5}{185.5}\right)$$

$$\alpha = \tan^{-1}(0.353099)$$

$$\alpha = 19.448112^{o}$$

Para el cálculo del *ángulo de visión con respecto al eje "y"*, también se plantea un triángulo rectángulo con las distancias que van del centro del área de visión hacia la cámara web (cateto adyacente) y del centro del área de visión al límite del área de operación del robot móvil con respecto al eje "y" (cateto opuesto).

A continuación, se utiliza la identidad trigonométrica *tangente* y se obtiene el *ángulo de visión con respecto al eje "y"* (Figura 5.7):

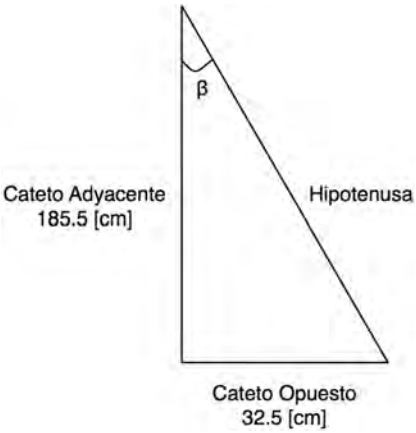

Figura 5.7 Triángulo formado para calcular el ángulo de visión de la cámara con respecto al eje y

$$\tan\beta = \frac{c.o.}{c.a.}$$

$$\beta = \tan^{-1}(\frac{c.o.}{c.a.})$$

$$\beta = \tan^{-1}(\frac{32.5}{185.5})$$

$$\beta = \tan^{-1}(0.175202)$$

$$\beta = 9.937483^{o}$$

Por lo tanto:

El ángulo de visión del dispositivo de adquisición de imágenes (cámara web) a la altura de 185.5 [cm] con respecto al eje “x” es ***19.4º*** y el ángulo de visión con respecto al eje “y” es ***9.9º*** .

5.3 Retraso de respuesta.

Es muy importante mencionar que las muestras experimentales y el valor complementario para calcular la distancia del objeto en cada cuadrante son afectados directamente por el retraso en el tiempo de procesamiento, envío y recepción de datos al robot móvil, el retraso en la respuesta de los componentes del robot móvil (servomotores y microcontrolador) y también por el retraso en la adquisición y procesamiento de las imágenes por parte del dispositivo de adquisición de imágenes (cámara web).

5.4 Soluciones al problema.

Planteada la problemática de operación que presenta el sistema, resulta necesario realizar ajustes en el software que adecuen el funcionamiento general, tomando en cuenta los factores que afectan directamente a la arquitectura anteriormente descrita. En la Figura 5.8 se presenta el ajuste en el software que solucionó la problemática de operación y en seguida se explicará cada sección:

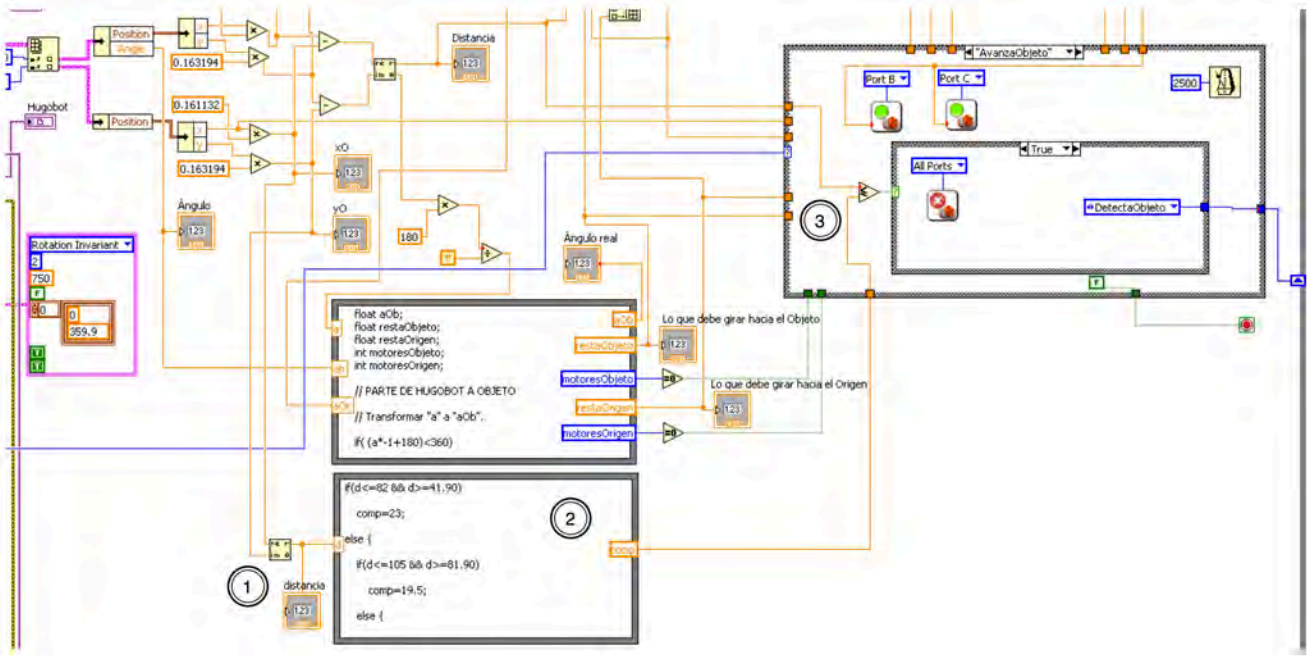

Figura 5.8 Ajuste de software en la implementación base

Sección 1.- Está ubicada en la etapa de procesamiento de datos. Se hace una conversión geométrica de las coordenadas cartesianas del objeto (*x,y*) a coordenadas polares (*r,θ*). La forma de hacer dicha conversión es usando las siguientes ecuaciones:

$$r = \sqrt{x^2 + y^2}$$

$$\theta = ang\tan(\frac{y}{x})$$

Esta sección se encarga ede proporcionar la distancia existente del objeto con respecto al origen, que es el dato de entrada de la siguiente sección.

Sección 2.- Igual que la sección anterior, está ubicada en la etapa de procesamiento de datos. Está contenida en un nodo de fórmula. De los valores mostrados en la tabla 5.2 se han tomado cuatro de ellos como parte de la solución, debido a que con estos valores se cubre la gran mayoría del área de operación del robot móvil. Estos valores son 14, 16, 19.5 y 23. A continuación, se presenta el código:

```
if(d<=82 && d>=41.90)      // Compara si el valor recibido está entre 82 y 41.90
   comp=23;                    // De ser cierto, la variable comp será 23
```

```
else {

    if(d<=105 && d>=81.90)   // Compara si el valor recibido está entre 105 y 81.90
      comp=19.5;                    // De ser cierto, la variable comp será 19.5
    else {
      if(d<=141 && d>=104.90)  // Compara si el valor recibido está entre 141 y 104.90
        comp=16;                    // De ser cierto, la variable comp será 16
      else {
        if(d<=157 && d>=104.90) // Compara si el valor recibido está entre 157 y 104.90
          comp=14;                    // De ser cierto, la variable comp será 14
        else
          comp=0;                     // En cualquier otro caso, comp será 0
      }
    }
}
```

Sección 3.- Esta sección está ubicada en la etapa de control, es decir, está en la máquina de estados en el estado "*AvanzaObjeto*". Recibe como entrada, el valor de la variable "comp", el cuál es el complemento que ayuda a calcular la distancia en perspectiva del objeto y la salida de esta sección puede ser cualquier valor entre 14, 16, 19.5 y 23, que es el valor usado para que la sujeción del objeto sea más confiable sin importar en qué área se encuentre.

Por lo tanto, el área de operación del robot móvil se reduce a la mostrada en la Figura 5.9:

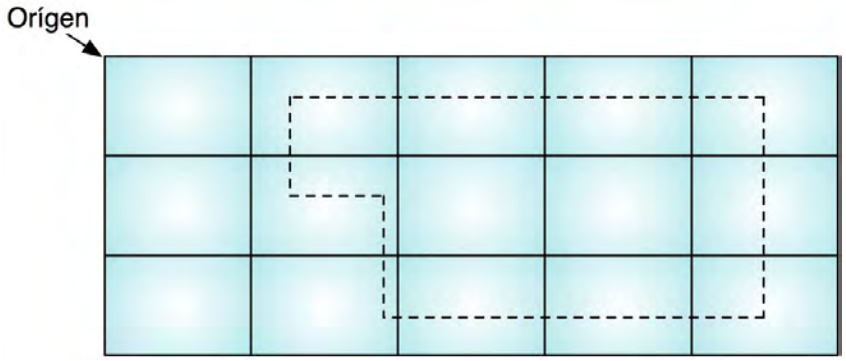

Figura 5.9 Área real de operación del robot móvil

5.5 Conclusiones.

El objetivo de la presente tesis logró cumplirse satisfactoriamente ya que se logró controlar el prototipo de un robot móvil aplicando la teoría de control difuso en conjunto con una máquina de estados, y el reconocimiento de patrones que forma parte del campo de la visión artificial en un entorno de programación LabVIEW.

El sistema funciona correctamente dentro de sus límites de operación. Estos límites están determinados en gran medida por el dispositivo de adquisición de imágenes, es decir, la cámara web, la cual no es la más óptima para el uso que se le dio en este trabajo.

Una mejora sustancial que modificaría ampliamente el comportamiento del sistema, sería utilizar una cám ara con mayor capacidad visual y con las siguientes características:

- 60 cuadros por segundo (FPS).
- Visión embebida.
- Escaneo progresivo.
- Sensor de imágenes CCD progresivo.
- Tiempo máximo de exposición 36.28 µs.

Con esta hipotética cámara, es posible aumentar la precisión en el comportamiento del robot móvil, ya que se tendrían 60 cálculos en un segundo, es decir, el 100% más de los cálculos por segundo actuales, esto implicaría una mayor rapidez en la respuesta del robot móvil.

La velocidad del desplazamiento del robot móvil podría aumentar drásticamente; la alineación del robot móvil con respecto al objeto y al origen sería en un período de tiempo mucho más breve al actual. Las variables de salida del controlador difuso “*errorAnguloObjeto*” y “*errorAnguloOrigen*”, los estados “*DetectaObjeto*” y “RegresaUnPoco” de la máquina de estados podrían desaparecer debido a que estas variables y estados fueron creados para compensar el retraso y la imprecisión provocados por la velocidad de adquisición actual de las imágenes. Esto provocaría también una reducción en el tiempo de ejecución de la rutina de recolección del robot móvil en un área mayor de operación y menor consumo de energía.

Bibliografía.

• Martín del Brio, Bonifacio, Sanz, Alfredo. Redes Neuronales y Sistemas Borrosos. Introducción teórica y práctica. RA-MA Editorial, Madrid, España, 1997.

• Wang, Li-Xin. A Course in Fuzzy Systems and Control. Prentice Hall PTR, United States of America, 1997.

• R. Alavala, Chennakesava. Fuzzy Logic and Neural Networks. Basic Concepts & Applications. New Age International Publishers, 2000.

• Takana, Kazuo, Wang, Hua O. Fuzzy Control Systems Desing and Analysis: A Linear Matrix Inequality Approach. John Wiley & Sons, Inc., 2001.

• Vicente, Pedro J., Albaladejo, José. Diseño Lógico. Universidad Politécnica de Valencia, Servicio de Publicaciones, Valencia, España, 1995.

• Scholz, Matthias P. Advanced NXT: The Da Vinci Inventions Book. Apress, 2007.

• Trobaugh, James J. Winning Design! Lego Mindstorms NXT Design Patterns for Fun and Competition. Apress, 2010.

• Valk, Laurens. The Lego Mindstorms NTX 2.0 Discovery Book. No starch Press, 2010.

Mesografía.

• **http://www.ni.com/vision/esa/vdm.htm**
Módulo NI Vision Development.
Diciembre 2011.

• **http://www.amcaonline.org.ar/ojs/index.php/mc/article/viewFile/3473/3390**
Prieto, Carlos E., Febres, Jesús E., Cerrolaza, Miguel, Miquelarena, Rodolfo. “Sistema de Visión Artificial para el Control de Movimiento de un Asistente Robótico Médico”.
Diciembre 2011.

• **http://www.intechopen.com/source/pdfs/17632/InTech-Digital_image_processing_using_labview.pdf**
Posada-Gómez,Rubén, Sandoval-González, Oscar O., Martínez, Albino Portillo-Rodríguez, Otniel, Alor-Hernández, Giner. “Digital Image Processing Using LabView”.
Diciembre 2011.

- **http://www.ni.com/academic/mindstorms/esa/**
  Lego Mindstorms NXT - LabVIEW.
  Noviembre 2011.

- **http://zone.ni.com/devzone/cda/tut/p/id/4435**
  LabVIEW Add-ons for Lego Mindstorms NXT.
  Noviembre 2011.

- **http://mindstorms.lego.com/en-us/Default.aspx**
  Noviembre 2011.

- **http://www.ni.com/pdf/manuals/372192d.pdf**
  PID and Fuzzy Logic Toolkit User Manual.
  Octubre 2011.

- **http://catarina.udlap.mx/u_dl_a/tales/documentos/lmt/ramirez_r_o/capitulo3.pdf**
  Ramírez, R. O. “CAPÍTULO 3 Lógica difusa”.
  Octubre 2011.

- **http://ants.dif.um.es/staff/juanbot/ml/files/20022003/fuzzy.pdf**
  Botía, Juan A. “Sistemas Difusos”.
  Octubre 2011.