99
29.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DETECCIÓN Y SEGUIMIENTO DEL MOVIMIENTO DE UN OBJETO EMPLEANDO PROCESAMIENTO DIGITAL DE IMÁGENES Y LÓGICA BORROSA

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN
P R E S E N T A

JORGE VALERIANO ASSEM.

DIRECTOR DE TESIS: M. en I. JOSÉ CASTILLO HERNÁNDEZ

LABORATORIO DE ELECTRÓNICA
CENTRO DE INSTRUMENTOS

MÉXICO, D.F. 1998

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Al hombre mas justo
del mundo ...
mi Padre.

A la mujer con carácter
inigualable ...
mi Madre.

A la mujer más cariñosa,
noble, responsable y
entregada a sus hijos ...
mi Madre.

A una joven brillante quien
sin duda será la mejor
Jurista de México ...
mi Hermana.

A quienes siempre han estado
presentes brindando su apoyo ...
mis Tíos José, María y
mis Primas Gabriela y Lucero.

A mis maestros ...
por sus conocimientos y
experiencias compartidas.

AGRADECIMIENTOS

En un día trascendente en la vida de todo universitario, como lo es su examen profesional, arriva un manantial de sentimientos, ilusiones y perspectivas, dentro de las cuales destaca la gratitud a tres pilares fundamentales en mi desarrollo profesional y como ser humano:

Mi familia, por estar siempre presente y apoyarme en mis aciertos y errores.

Mi alma mater, la Universidad Nacional Autónoma de México, especialmente la Facultad de Ingeniería, por su nobleza, generosidad y dignidad.

Al Centro de Instrumentos, en particular a los Laboratorios de Electrónica y Neurocontrol, por el apoyo para desarrollar este proyecto.

Así mismo agradezco a:

M.I. José Castillo Hernández la asesoría y calidad humana brindada en la realización de mi tesis.

M.C. José Luis Pérez Silva por permitirme formar parte del Laboratorio de Electrónica.

M.I. Miguel Angel Bañuelos Saucedo, M.I. Wilfredo Martínez Payán, Ing. Gerardo Venancio Calva Olmos, Ing. Rosendo Fuentes González, Fís. Alejandro Padrón Godínez, por los comentarios, sugerencias y amistad brindados durante esta investigación.

Mis sinodales Ing. Pablo García y Colomé, M.C. José Luis Pérez Silva, M.I. Miguel Angel Bañuelos Saucedo, M.I. Wilfredo Martínez Payán.

Mi hermana María Araceli Valeriano Assem por el tiempo que invirtió en la revisión de estilo.

Mis amigos del Centro de Instrumentos.

Mis amigos Saeid Akhavan Tajeri, Jorge Alberto Andrés Aguilar, Eduardo de Jesús Arellano Moreno, Gabriel Fernández Nuñez, Arturo González Esquivel, Verenice Gurrola Herrera, Mónica Salmerón Arteaga, Antonio Sánchez García, Francisco Valdez Souto por su amistad incondicional.

ÍNDICE

INTRODUCCIÓN.

Una de las lineas de trabajo del Laboratorio de Electrónica y Neurocontrol del Centro de Instrumentos de la Universidad Nacional Autónoma de México, es el área de *robótica*. Un problema de interés es el control del robots empleando retroalimentación visual basada en el reconocimiento de patrones. El presente trabajo aborda el uso de retroalimentación visual para el seguimiento de un objeto en movimiento sobre un fondo uniforme, como una base para el control utilizando reconocimiento de patrones.

El concepto de visión por computadora, conocido comúnmente como "*Machine Vision*" o "*Computer Vision*", se define de varias maneras pero en esencia, todas coinciden en lo mismo. Dos definiciones obtenidas de las referencias [1] y [2] son las siguientes:

- "Es la empresa de automatizar e integrar una amplia variedad de procesos y representaciones usando percepción visual".

- "El proceso de extraer, caracterizar e interpretar información de imágenes provenientes del mundo tridimensional".

Con base en lo anterior y desde nuestro punto de vista llamamos visión por computadora a *"un proceso de adquisición, procesamiento e interpretación de imágenes con base en el cual es posible tomar decisiones de acción para controlar automáticamente algún proceso"*.

Visión por computadora representa una herramienta útil en diversas ramas del conocimiento y de la industria. Gran parte de las decisiones tomadas por el hombre en el control de procesos se hacen a través del sentido de la vista. Por tanto, un dispositivo capaz de tomar decisiones con base en información visual y, de manera confiable puede hacerse cargo de tales tareas, sobre todo en actividades peligrosas para la vida del hombre.

Las áreas de aplicación son variadas y crecientes; por ejemplo, en la referencia [3] se describe una retroalimentación visual para controlar un robot que reconoce y recoge piezas sobre una banda en movimiento, en la referencia [4] se describe un sistema visual para planeación, seguimiento y minimización de colisiones en robots, en la referencia [5] se describe el control del movimiento de un robot usando retroalimentación visual. Algunas otras áreas de aplicación mencionadas en la referencia [1] abarcan el uso de imágenes por el ejército para la guía de misiles, en astronomía para determinar la composición química de las estrellas, en medicina para diagnosticar anormalidades en los órganos del cuerpo humano, etc.

Contar el número de aviones militares localizados en una pista utilizando las imágenes proveniente de un satélite, reconocer el rostro de una persona dentro de un conjunto de imágenes etc. son ejemplos complejos donde se puede aplicar visión por computadora. El problema de visión es no trivial, puede ir desde calcular el centroide de una imagen utilizando detección de bordes, pasando por el problema de no uniformidad en el fondo de la imagen así como de ruido incorporado por el sistema de adquisición y, hasta el *reconocimiento* de objetos inmersos en un ambiente con muchos más objetos.

Existen tres niveles en los cuales es posible hacer visión:

- *Nivel alto.* Involucra capacidades de conocimiento que pueden guiar actividades visuales y en consecuencia, un sistema de visión debe poder tomar ventaja de ello. Por ejemplo, tareas como cruzar una avenida muy transitada, llegar a una fiesta y "ver" rápidamente ¿a quién se conoce?, ¿quién es el anfitrión? etc. son tareas que requieren gran cantidad de conocimiento de los objetos que vemos en el mundo y por ende, son capacidades de nivel alto.

- *Nivel bajo.* Involucra capacidades asociadas a procesos primitivos que pueden considerarse como "reacciones automáticas", no requiriendo inteligencia en la parte del sistema de visión. Estas capacidades pueden ser equiparables al proceso de sensado y preprocesamiento de las imágenes. Una importante capacidad de bajo nivel es la percepción del objeto.

- *Nivel medio.* Involucra capacidades asociadas a procesos que extraen características y etiquetan componentes de una imagen resultado de una visión de nivel bajo.

El papel de las computadoras en la implementación de sistemas de visión es importante debido a que:

- Son versátiles, fácilmente reconfigurables y su trabajo puede ser escudriñado hasta el más fino detalle.

- Los procesos y entidades de la ciencia de la computación proveen poderosas herramientas conceptuales para trabajar en percepción y conocimiento.

- Pueden dar medídas precisas en el procesamiento que hacen.

- Los modelos desarrollados en computadora pueden ser evaluados por su eficacia, por su organización interna, por su estructura etc.

El uso de una computadora para el proceso de visión no es requisito; no obstante, ésta se emplea por las facilidades que brinda; es más sencillo programar una rutina de procesamiento en una computadora que su implantación en hardware.

Planteamiento.

Ubicada la magnitud de un problema de visión por computadora; en el presente trabajo se desea crear un sistema de visión que "*Permita seguir objetos en movimiento, logrando que estos permanezcan siempre en el campo de visión de una cámara de video CCD*".

Delimitación del problema.

El planteamiento determinado para el sistema de visión es amplio, debido a que abarca toda la gama de subproblemas antes mencionados; a continuación se dan las limitaciones que debe satisfacer:

- Seguimiento de sólo un objeto a la vez sobre un fondo uniforme.
- Reducción de la influencia de ruido incorporado por el ambiente y el sistema de adquisición de datos.
- Funcionamiento orientado a manipular y procesar 30 cuadros por segundo (estándar NTSC).
- Algoritmos de procesamiento programados en una PC.

Análisis.

Nuestro sistema de visión ilustrado en la Fig. A, está conformado por tres subsistemas; a saber:

- *Adquisición de imágenes de video.* Está constituido por una cámara de video CCD y, una tarjeta digitalizadora, la cual, se encarga de adquirir las imágenes empleadas por un algoritmo de procesamiento y control para el seguimiento del objeto. Este subsistema debe ser capaz de capturar y transferir imágenes a 30 cuadros por segundo (estándar NTSC) al sistema de memoria principal de la computadora, así como permitir accesar la información de los pixeles de la imagen.

- *Procesamiento digital y control para el seguimiento del objeto.* Constituido por una computadora personal, es la parte medular del sistema de visión porque aquí se realiza el procesamiento de las imágenes adquiridas y el algoritmo de control para el seguimiento del objeto. De este subsistema depende la susceptibilidad al ruido y la aproximación que se tenga de un procesamiento en tiempo real.

- *Estructura mecánica y electrónica para la ejecución del movimiento.* Está encargada de dar movimiento a la cámara de video CCD a través de una estructura mecánica, desplazada mediante motores controlados por una electrónica; la cual, ejecuta la acción de control calculada por el algoritmo para el seguimiento del objeto.

Desarrollo e implantación de la solución.

El presente trabajo está constituido por cuatro capítulos y dos apéndices que conforman el desarrollo de soluciones e implantación final, como se describe a continuación:

Capítulo 1. Revisa los antecedentes necesarios para el desarrollo de la solución en dos direcciones básicas. La primera, aborda conceptos de Lógica Borrosa aplicada principalmente a sistemas de Control Lógico Borroso, estudiando sus cuatro elementos fundamentales: base de reglas borrosas, interface de fusificación, maquina de inferencias e, interface de defusificación. La segunda, expone conceptos utilizados en Procesamiento Digital de Imágenes orientados a detección de bordes. Se exploran las siguientes tres técnicas de filtrado: *Usando operador Laplace, Lógica Borrosa y Comparación de Mínima Distancia.*

Capítulo 2. Desarrolla un simulador que permite, en un entorno lo más parecido al sistema real, evaluar los algoritmos para la detección y seguimiento del objeto en movimiento. Simula el movimiento del objeto sobre una trayectoria y, la adquisición y movimiento de la cámara de video.

Usando el simulador se evalúa, en primer lugar, el desempeño de las técnicas para la detección de bordes y extracción de la posición del objeto; en segundo lugar, se diseñan y evalúan dos sistemas de control para el seguimiento del objeto, el primero efectúa una Corrección Simple del Error (CSE), mientras el segundo es un Controlador Lógico Borroso (CLB).

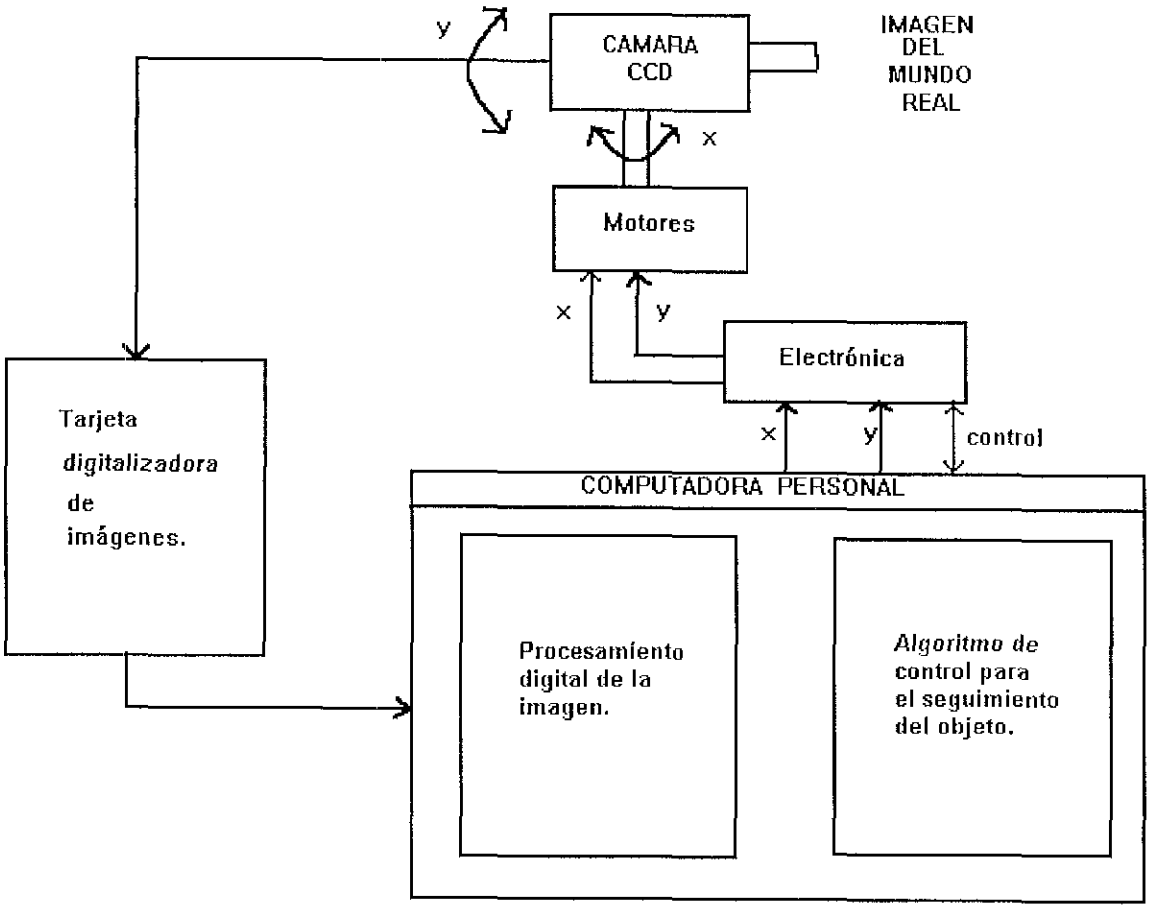


Fig. A. Sistema de visión.

Capítulo 3. Aborda la implantación del sistema físico con las técnicas estudiadas en el simulador. Describe el sistema de adquisición de imágenes utilizado (cámara y tarjeta digitalizadora), diseño y construcción en circuito impreso de la electrónica para el movimiento de motores, estructura mecánica para el soporte y movilidad de la cámara y, software para la conjunción del sistema físico global que controla los tres subsistemas anteriores.

Apéndice A. Presenta el diccionario de datos y código en C++ del simulador. Incluye código para todas las técnicas utilizadas de Procesamiento de Imágenes, así como el sistema de Control Difuso diseñado.

Apéndice B. Contiene el diccionario de datos y código en C++ para la conjunción del sistema físico. Comprende los procedimientos para adquisición de imágenes a través de la tarjeta digitalizadora, inicialización y control de la electrónica para el movimiento de motores, así como las técnicas óptimas evaluadas en el simulador para la extracción de bordes, posición y seguimiento del objeto en movimiento.

CAPÍTULO 1
ANTECEDENTES.

Este capítulo presenta las herramientas teóricas utilizadas, en primer lugar, para hacer el procesamiento digital de la imagen adquirida y, en segundo para el seguimiento del objeto. La primera estudia tres técnicas de Reconocimiento de Bordes y, la segunda un Sistema de Control basado en Lógica Borrosa.

1.1 Lógica y Control Borroso.

Los antecedentes de Lógica y Control Borroso tratados en esta sección están fundamentados en las referencia [6] y [12].

El concepto de Conjuntos Borrosos, se atribuye directamente a L.A. Zadeh en su trabajo publicado en 1965 en la referencia [7]. El propósito original de Zadeh al introducir los conjuntos borrosos fue proveer de una herramienta para ayudar en el modelado de fenómenos complejos, especialmente, pero no restringido, hacia aquéllos que involucran agentes humanos.

En consecuencia, la Lógica Borrosa y en especial su aplicación en control, ha mostrado ser una alternativa para procesos donde el control de la planta es tal, que la concepción de un modelo matemático de su comportamiento es complejo. Ejemplos en donde se decidió usar control borroso son: Industria Automotriz en la referencia [8], Equipo para Procesamiento de Imágenes en la referencia [9] etc.

La *Lógica Borrosa* permite manejar el concepto de grados de verdad o falsedad y no solamente la verdad o falsedad absoluta como es el caso de la *Lógica Booleana.* Debido a esta característica de flexibilidad es que se aplica como una alternativa para controlar sistemas no lineales.

1.1.1 Conjuntos Borrosos.

Para conjuntos clásicos, la transición entre pertenencia y no-pertenencia de un elemento a un conjunto es abrupta y bien definida: un elemento simplemente *pertenece* o *no pertenece* a un conjunto.

Subconjunto Clásico: Un subconjunto clásico $A(x)$ de un universo X es una colección de elementos que cumplen con ciertas características; por ejemplo, $A(x) \subseteq X$, donde X es el conjunto de los números enteros y $A(x)$ el subconjunto de los números menores o iguales a 10; en este caso, los números 1, 2, 3 y 4 pertenecen al subconjunto $A(x)$, los números 11, 12 y 13 no pertenecen y los números 1.5, 2.1 y 3.3 aunque son menores a 10 no pertenecen al subconjunto $A(x)$, porque no están definidos en el universo. Un subconjunto clásico es caracterizado por una función $\mu: x \rightarrow \{1,0\}$ que asocia a cada elemento de X uno de dos posibles valores {0,1}, donde el 0 indica la no pertenencia del elemento al subconjunto y el 1 la pertenencia al mismo; es decir, no existe incertidumbre para decidir si es miembro o no del conjunto.

Para conjuntos borrosos, la transición entre la pertenencia o no pertenencia de un elemento a un conjunto puede ser gradual; este es el principio fundamental de la *Lógica Borrosa.*

Subconjunto Borroso: En este caso la pertenencia del elemento al conjunto no es absoluta; es decir, existe una incertidumbre. La caracterización del conjunto está definida por una función $\mu: x \rightarrow [1,0]$ que asocia a cada elemento de X con un número real en el intervalo [0,1]; a dicha función se le conoce como *función de membresía.* Es común asociar a cada conjunto una etiqueta o término lingüístico borroso. Con base en lo anterior un subconjunto borroso lo representaremos como una colección de pares ordenados $(x_i, \mu(x_i))$ es decir:

$$A(x) = \int_X \frac{\mu_A(x_i)}{x_i}$$

donde A es un término lingüístico con el cual se identifica al subconjunto borroso, el signo de integral es usado para denotar una colección de elementos y el signo de división "/" no indica una operación aritmética sino la correspondencia que existe entre el elemento x_i y el grado de membresía $\mu_A(x_i)$.

La Fig. 1.1 muestra las funciones de membresía posibles para conjuntos *clásicos* y conjuntos *borrosos*. Nótese que la función de membresía *clásica* representa un cambio abrupto entre la pertenencia y no pertenencia, mientras la función de membresía *borrosa* puede ser cualquiera dentro del intervalo de pertenencia [0, 1].

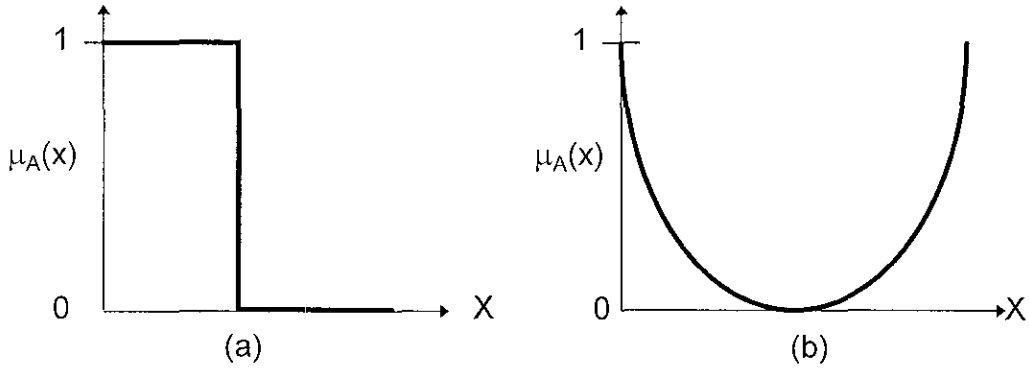


Fig. 1.1 (a) Función de membresía para un conjunto clásico A. (b) Función de membresía para un conjunto borroso A.

Existen diversos tipos de funciones de membresía, las más comunes son de tipo triangular o trapezoidal. La Fig. 1.2 muestra dos ejemplos de funciones de membresía, donde el eje de las abscisas representa el universo del discurso *X* y el eje de ordenadas los grado de membresía $\mu_A(x)$ para cada elemento de *X*.

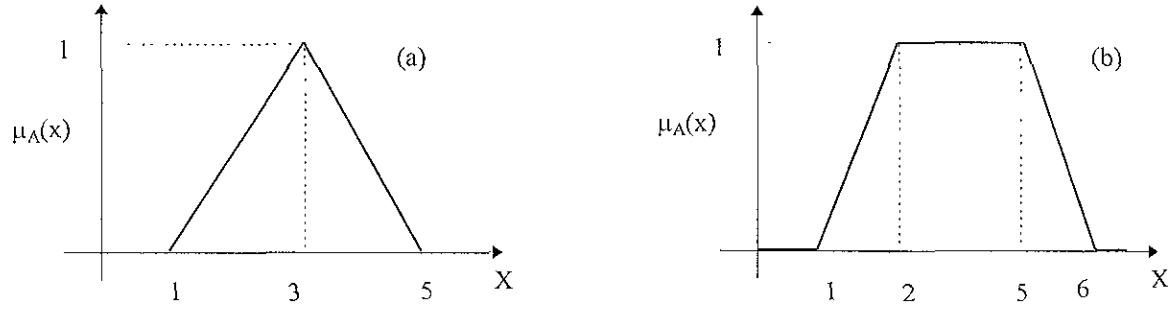


Fig. 1.2. (a) Función de membresía triangular, (b) Función de membresía trapezoidal.

Como un ejemplo de conjuntos borrosos vamos a hablar sobre la "*estatura*" de las personas. En este caso el universo del discurso es el conjunto de personas. Definiremos el subconjunto borroso *ALTO*, el cual responderá a la pregunta ¿ Qué tan alta es la persona x ?. *ALTO* es una *variable lingüística*; la cual, representa nuestra categoría cognoscitiva de "*estatura*". A cada persona en el universo del discurso debemos asignarle un grado de pertenencia al conjunto borroso *ALTO*. La manera más fácil de hacerlo, es usando una función de membresía basada en la estatura de la persona:

$$ALTO(x) = \begin{cases} 0 & Si\ estatura(x) \leq 1.50\ m \\ [estatura(x) - 1.50] / 0.30 & Si\ 1.50 < estatura(x) < 1.80 \\ 1 & Si\ estatura(x) \geq 1.80\ m \end{cases}$$

Dada esta definición, si Erik tiene una estatura de *estatura(Erik)=1.75 m,* entonces el grado con que Erik pertenece al conjunto borroso *ALTO* es *ALTO(Erik)=0.8333.* Expresiones como "A es X" pueden ser interpretadas como grados de verdad. Ejemplo: "Erik es ALTO" = 0.8333.

1.1.2 Lógica Borrosa.

El tipo de lógica borrosa, queda establecida cuando se define cuál es la forma de los operadores T-norma (conjunción), T-conorma (disyunción) y negación, así como una función de implicación. La Tabla 1.1 resume la definición más común de estos operadores, donde sus argumentos corresponden a grados de pertenencia. Es importante notar que si se tuvieran exclusivamente valores de pertenencia "0" y "1" en las definiciones de la Tabla 1.1, se obtienen las funciones booleanas convencionales.

Tabla 1.1

Operador	Símbolo	Definición
T-norma	$\wedge$	$x \wedge y$ = mínimo(x,y)
T-conorma	$\vee$	$x \vee y$ = máximo(x,y)
Negación	$\neg$	$\neg x = 1-x$

Tabla 1.1. Definiciones más comunes de los operadores borrosos.

Como ejemplo de las operaciones borrosas, vamos a definir una nueva función de membresía que describe al conjunto borroso *VIEJO*:

$$VIEJO(x) = \begin{cases} 0 & Si\ edad(x) \leq 18\ años \\ [edad(x) - 18] / 42 & Si\ 18 < edad(x) < 60\ año \\ 1 & Si\ edad(x) \geq 60\ años \end{cases}$$

Si consideramos que: edad(x)=45 años y estatura(x)=1.65 m, entonces:

x es *ALTO* y x es *VIEJO* = 0.5
x es *ALTO* o x es *VIEJO* = 0.642
$\neg$x es VIEJO = 0.358

En el caso de lógica borrosa, Zadeh en la referencia [10], indica: si A y B son dos subconjuntos borrosos entonces, en una declaración del tipo SI A

ENTONCES B, la implicación describe una relación entre los subconjuntos y, sugiere que la implicación bajo el contexto de subconjuntos borrosos se define por una relación borrosa, en lugar de conectivos, como es el caso de la lógica clásica.

Una relación entre dos subconjuntos borrosos $A(x)\subseteq X$ y $B(y)\subseteq Y$, donde $x\in X$ y $y\in Y$, es otro subconjunto borroso R(x,y), que se genera a partir del producto cartesiano $A(x)\times B(y)$. Este subconjunto es caracterizado por una función de membresía $\mu_R(x,y)$. Por tanto, una relación borrosa se expresa como:

$$R(x,y)=\int_{X\times Y}\frac{\mu_R(x_i,y_j)}{(x_i,y_j)}$$

El tipo de implicación se define por la forma en que se obtiene la función de membresía $\mu_R(x,y)$. Existen diversas funciones de implicación, la más común es la función de implicación Mamdani conocida como MIN, definida como I(x,y)=mínimo(x,y). A continuación daremos un ejemplo de una relación borrosa.

Sea el subconjunto $A(x)=\{0.1/1,\ 0.2/2,\ 0.5/3,\ 0.8/4,\ 1/5\}$ y el subconjunto $B(x)=\{0.2/1,\ 0.4/2,\ 0.8/3,\ 1/4,\ 0.6/5\}$. Considerando la función de implicación de Mamdani, la relación borrosa $R(x,y)$ de los conjuntos $A(x)$ y $B(y)$ es:

$$R(x,y)=\int_{X\times Y}\frac{MIN(\mu_A(x_i),\mu_B(y_i))}{(x_i,y_j)}$$

$$R(x,y)=\left\{\frac{0.1}{(1,1)},\frac{0.1}{(1,2)},\frac{0.1}{(1,3)},\frac{0.1}{(1,4)},\frac{0.1}{(1,5)},\frac{0.2}{(2,1)},\frac{0.2}{(2,2)},\frac{0.2}{(2,3)},\frac{0.2}{(2,4)},\frac{0.2}{(2,5)},\frac{0.2}{(3,1)},\frac{0.4}{(3,2)}\right.$$
$$\left.\frac{0.5}{(3,3)},\frac{0.5}{(3,4)},\frac{0.5}{(3,5)},\frac{0.2}{(4,1)},\frac{0.4}{(4,2)},\frac{0.8}{(4,3)},\frac{0.8}{(4,4)},\frac{0.6}{(4,5)},\frac{0.2}{(5,1)},\frac{0.4}{(5,2)},\frac{0.8}{(5,3)},\frac{1}{(5,4)},\frac{0.6}{(5,5)}\right\}$$

Esta colección de pares ordenados con sus correspondientes grados de membresía, puede representarse en forma de una matriz:

	1	2	3	4	5
1	0.1	0.1	0.1	0.1	0.1
2	0.2	0.2	0.2	0.2	0.2
3	0.2	0.4	0.5	0.5	0.5
4	0.2	0.4	0.8	0.8	0.6
5	0.2	0.4	0.8	1	0.6

donde el renglón representa los componente del conjunto *A(x)* y la columna corresponde a los componentes del conjunto *B(y)*.

1.1.3 Razonamiento en un Controlador Lógico Borroso (CLB).

El razonamiento en un Controlador Lógico Borroso se compone de cuatro elementos principales:

- una base de reglas borrosa.
- una interface de fusificación.
- una máquina de inferencias.
- una interface de defusificación.

1.1.3.1.- Base de reglas.

Es un conjunto de declaraciones lingüísticas del tipo *SI... ENTONCES...* con vaguedad en su predicado. Se deriva a partir de conocimiento, intuición y experiencia del comportamiento del sistema que se desea controlar. Estas reglas describen cual es la relación entre la entrada y salida del controlador. La parte *SI...* se conoce como antecedente y contiene un conjunto de condiciones; la parte *ENTONCES...* es llamado consecuente y contiene la conclusión de la regla. *SI* el antecedente se satisface, *ENTONCES* el consecuente se aplica. En el caso de un sistema de control lógico borroso de múltiples-entradas-una-salida, la base de reglas es la siguiente:

$$\begin{aligned} R_1&: \quad si\ X^1\ es\ A_1^1\ \ y\ .....\ y\ X^n\ es\ A_n^1\ entonces\ Z\ es\ B^1 \\ R_2&: \quad si\ X^1\ es\ A_1^2\ \ y\ .....\ y\ X^n\ es\ A_n^2\ entonces\ Z\ es\ B^2 \\ &\vdots \\ R_m&: \quad si\ X^1\ es\ A_1^m\ \ y\ .....\ y\ X^n\ es\ A_n^m\ entonces\ Z\ es\ B^m \end{aligned} \tag{1.1}$$

donde X^i *(i=1...n)* son las entradas al sistema de reglas, Z es la salida del controlador, A_j^i y B^j *(j=1...m)* son términos lingüísticos donde m es el número de reglas .

1.1.3.2 Fusificación

La interface de fusificación genera una función de membresía para los conjuntos borrosos de entrada del sistema. Específicamente si X es el universo del discurso de una variable de entrada "x", y $x=x_o \in X$ es el valor de la variable, la salida de la interface de fusificación es un subconjunto borroso $A'(x)=Fuzzy(x_o)$, en X. Uno de los principales métodos de fusificación es la *fusificación por punto*, definida como:

$$\mu_{A'(x)} = \begin{cases} 1 & si\ x = x_0 \\ 0 & Cualquier\ otro\ caso \end{cases}$$

Es decir, el subconjunto *A'(x)* es un subconjunto donde todos los elementos del universo *X* tienen un grado de pertenencia cero a excepción del elemento x_0, al cual, se le asigna el mayor grado de pertenencia al conjunto. Este conjunto también se le conoce como *singleton*.

Esta técnica de fusificación es ampliamente usada en aplicaciones de control borroso por su natural y fácil implementación; ya que, el proceso de fusificación se reduce a evaluar el valor de entrada en las funciones de membresía. En la Fig. 1.3 se da un ejemplo de este tipo de fusificación. En este el universo *X* es *{1,2,3,4,5}* y el valor que toma *x* es valor de 3; por tanto, el subconjunto $\mu_{A'}(x)$ que produce la fusificación será *{0/1, 0/2, 1/3, 0/4, 0/5}*.

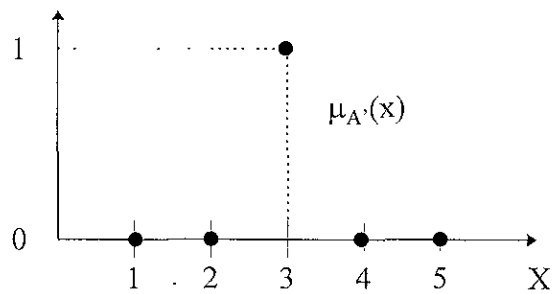


Fig. 1.3.- Resultado de una fusificación por punto

Otro método es la *fusificación aproximada*, donde el subconjunto *A'(x)* es un subconjunto borroso definido por:

$$\mu_{A'(x)} \neq 0 \qquad si\ |x - x_0| < \delta$$

En este caso el subconjunto A'(x) está formado por elementos cuyo grado de pertenencia es diferente a cero. Los elementos que cumplen con éstos, son todos aquéllos cuyo valor absoluto de la diferencia entre x y x_0 es menor al intervalo δ. La función de membresía asociada es comúnmente triangular o trapezoidal; sin embargo, no hay nada que la limite. Por ejemplo, sea *X* el universo de entrada definido por el intervalo continuo [0, 10]. Si la entrada es *x=4*, δ*=2* y la función de membresía es triangular, el conjunto borroso *A'(x)* generado por la fusificación, será igual al mostrado en la Fig. 1.4.

1.1.3.3 Máquina de inferencias.

Para realizar la inferencia (razonamiento) en un sistema de inferencia borroso, se efectúa la evaluación de cada regla seguida por la composición de éstas. Cada

método de composición refleja una máquina especial de inferencia. Algunos mecanismos de inferencia según la referencia [11] son:

- Min-Max. Al evaluar cada regla se utiliza el operador MIN y, para la agregación de reglas el operador MAX:

$$B'(z) = \underset{z \in Z}{Max} \left\{ Min(\mu_{Ai}(x), \mu_{Bi}(y)) \right\}$$

- Prod-Max. Al evaluar cada regla se utiliza el operador PROD y, para la agregación de reglas el operador MAX:

$$B'(z) = \underset{z \in Z}{Max} \left\{ (\mu_{Ai}(x) \bullet \mu_{Bi}(y)) \right\}$$

- Max-Min. $B'(z) = \underset{z \in Z}{Min} \left\{ Max(\mu_{Ai}(x), \mu_{Bi}(y)) \right\}$

- Max-Max. $B'(z) = \underset{z \in Z}{Max} \left\{ Max(\mu_{Ai}(x), \mu_{Bi}(y)) \right\}$

- Min-Min. $B'(z) = \underset{z \in Z}{Min} \left\{ Min(\mu_{Ai}(x), \mu_{Bi}(y)) \right\}$

Los métodos más empleados para la evaluación de reglas son el Min-Max y el Prod-Max.

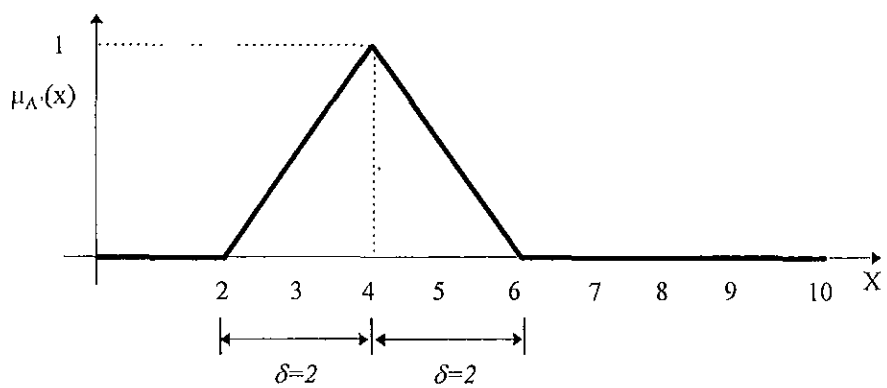


Fig. 1.4.- Resultado de una fusificación aproximada

1.1.3.4 Defusificación.

Con base en el conjunto de salida B'(z) se presenta el problema de elegir un valor particular "z" representativo del conjunto; la primera idea que surge es elegir el elemento "z" con mayor grado de pertenencia al conjunto; sin embargo, ¿qué ocurre cuando el conjunto final tiene más de un elemento que tenga asociado el máximo grado de membresía del conjunto?. Por ejemplo, en la Fig. 1.5 se muestra un conjunto borroso de salida, ¿cuál de los valores debemos elegir 1.10 ó 2.05?, ¿una interpolación sería adecuada? ó ¿el valor medio entre ambos?.

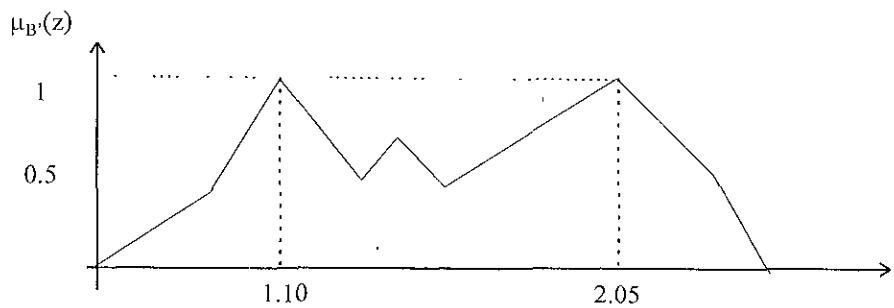


Fig. 1.5.- Función de membresía generada por la máquina de inferencias

Por otro lado, el elemento representativo del conjunto no necesariamente debe ser aquel con el mayor grado de pertenencia, por ejemplo la Fig. 1.6 muestra otro conjunto borroso producido por una máquina de inferencia. En este caso el elemento 1.80 tiene el mayor grado de membresía, pero existen un número esencialmente mayor de elementos cuyos valores asociados son iguales entre sí y diferente del máximo. En este caso es de esperarse que el elemento representativo del conjunto tuviera asociado un valor de pertenencia si no igual, si muy próximo al que presenta la mayoría de los elementos del conjunto. Bajo este contexto se puede decir que la representación del subconjunto es una distribución de probabilidad y la defusificación es el proceso mediante el cual obtenemos el promedio de dicha distribución de probabilidades.

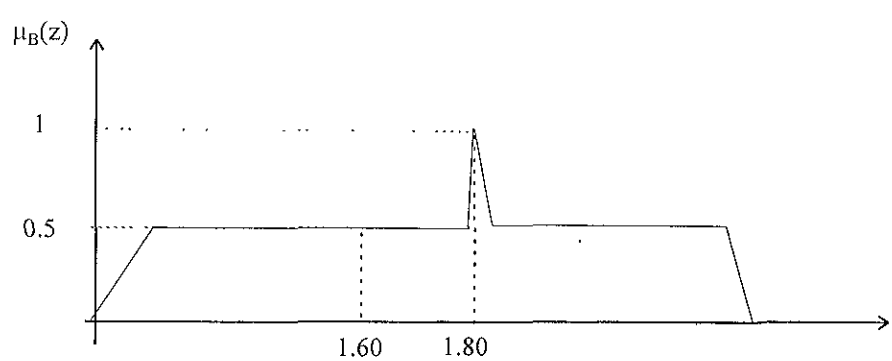


Fig. 1.6.- Función de membresía generada por la máquina de inferencias

Existen muchos métodos de defusificación y de ellos depende en gran medida el tiempo de respuesta del sistema de inferencia y el tiempo de obtención de la respuesta. Algunos de ellos son:

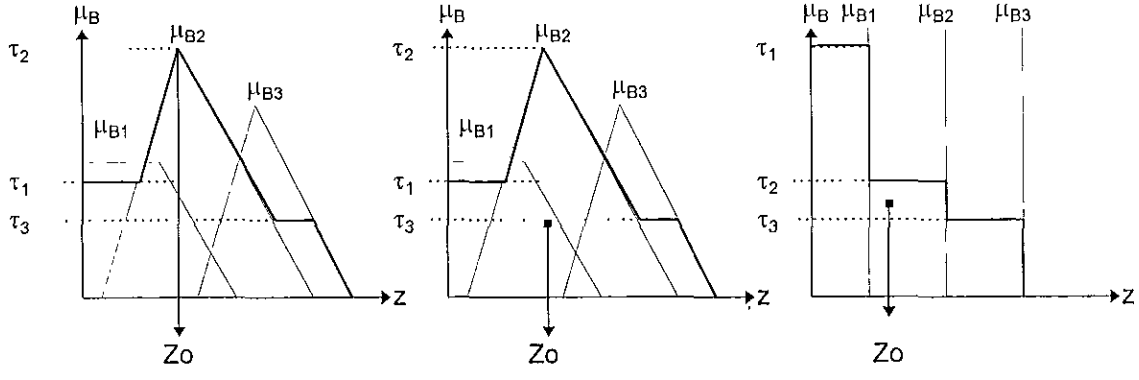
- *Método de la máxima pertenencia:* Se aplica cuando el área obtenida con la agregación de reglas tiene un sólo punto pico. El grado de membresía asociado al pico de ésta área se toma como el valor de salida del sistema de inferencia borroso. Esto se ilustra en la Fig. 1.7 (a).

- *Método de la media de los máximos:* El valor defusificado se determina como una media de todos los valores que tienen grados de pertenencia máximos (picos) en el área obtenida para la variable de salida con la agregación de reglas.

- *Método del centroide:* El valor que se toma como salida del sistema de inferencia borroso es el valor correspondiente al centro de gravedad del área obtenida con la agregación de reglas, como se ilustra en la Fig. 1.7 (b). A este método de defusificación en conjunción con el método de inferencia Min-Max se le conoce como *Método de Mamdani* en la referencia [12]. El centroide se obtiene con la expresión:

$$z_0 = \frac{\int_z \mu_{B'}(z).z}{\int_z \mu_{B'}(z)}$$

Existe una variación sencilla del método del centroide. Consiste en emplear funciones "*singleton*" para los conjuntos borrosos de salida; estas describen un conjunto borroso que consta de un sólo elemento. Con los grados de membresía obtenidos del proceso de inferencia se construye una área como la de la Fig. 1.7 (c) y el valor de salida se obtiene con el centroide de esa área.



(a) Máxima pertenencia *(b) Mamdani* *(c) Sugeno*

Fig. 1.7 Diversos métodos de defusificación.

Este método es muy utilizado por su bajo requerimiento en tiempo de cómputo en comparación con la integral. Este método de defusificación en conjunto con el método de inferencia Min-Max se conoce como *Método del Valor Verdadero (TVFI, Truth Value Flow Inference)* e la referencia [12], o bien, como *Método de Sugeno* en la referencia [13]. La expresión para obtener el valor defusificado es la siguiente:

$$z_0 = \frac{\sum_{j=1}^{w} \mu_{B'}(z_j).z_j}{\sum_{j=1}^{w} \mu_{B'}(z_j)}$$

donde *w* es el número de funciones de membresía Singleton utilizadas para la salida. La Fig. 1.8 ilustra gráficamente el algoritmo de razonamiento usando el método de inferencia Min-Max y una defusificación tipo *Sugeno.*

1.1.3.5 Controlador Lógico Borroso (CLB).

El diagrama básico para un sistema de control a lazo cerrado con base en la referencia [12] es ilustrado en la Fig. 1.9. La *Planta* (proceso, vehículo, etc.) a ser controlada se denomina *sistema* y se denota como "*S*". El propósito del controlador a lazo cerrado es garantizar una respuesta deseada para la salida "*y*". El objetivo es mantener la salida "*y*" lo más cercana posible al punto de referencia "*w*" en presencia de perturbaciones, fluctuaciones y ruido. La entrada "*e*" del controlador es el error de la salida con respecto a la referencia. La salida "*u*" del controlador es la acción de control aplicada a la planta.

Cuando se trata de control convencional, la ley de control se define con una o varias ecuaciones analíticas (algebráicas, diferenciales, en diferencias, ...) que son obtenidas a partir de un análisis matemático formal; siendo este uno de los paradigmas principales del control convencional.

En el caso del Control Lógico Borroso, la *ley de control se define por medio de un sistema de inferencia borroso;* es decir, la ley de control está contenida en una base de conocimiento formada por reglas borrosas del tipo SI... ENTONCES... con vaguedad en su predicado y un mecanismo de inferencia borrosa, como se explicó en las secciones anteriores y específicamente como se mostró en la ecuación (1.1). La diferencia fundamental de un CLB con respecto del control convencional, es el hecho de que no se emplea una descripción analítica del sistema. El paradigma principal en control borroso es, que el algoritmo de control es una base de conocimiento descrita por los métodos de la lógica borrosa.

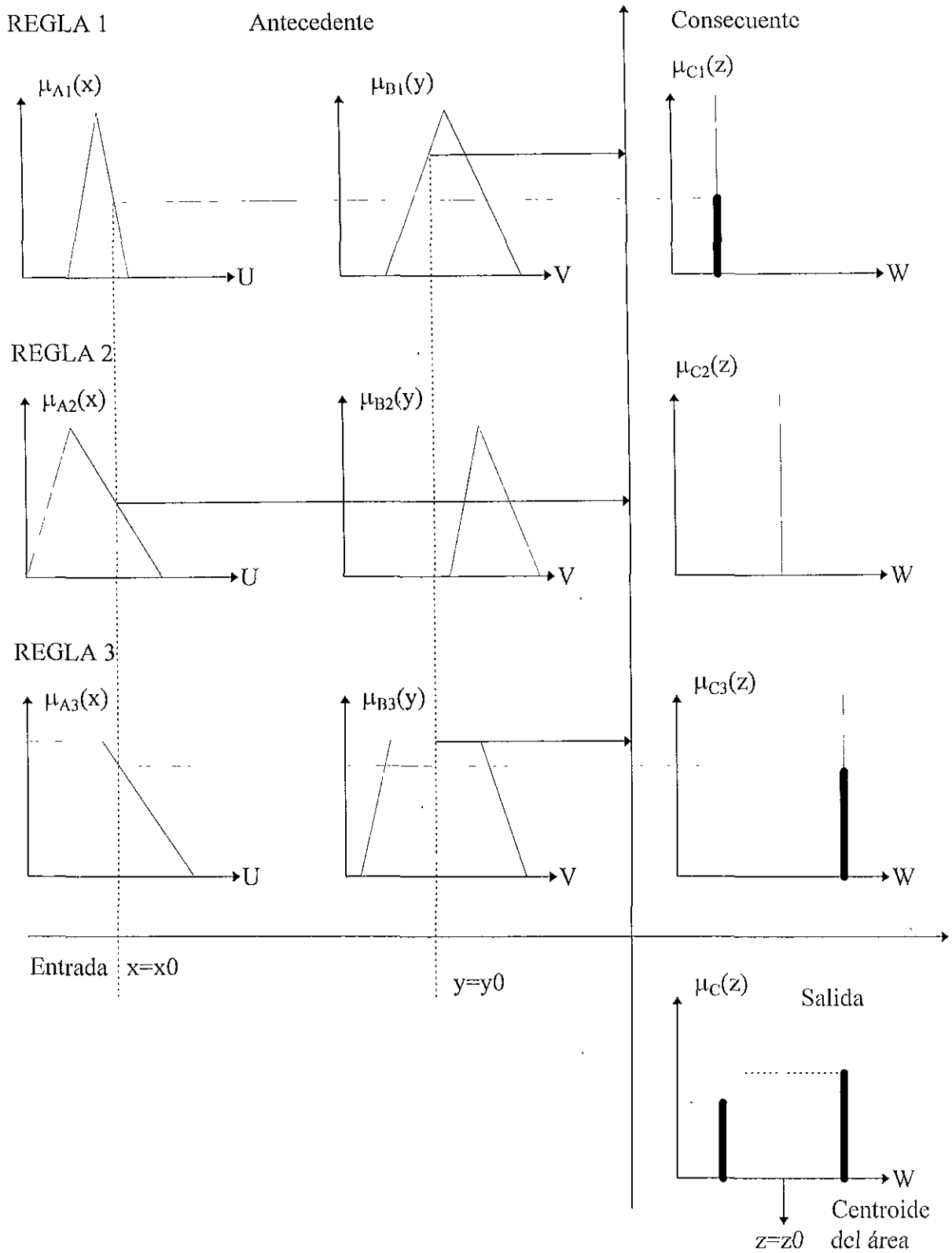


Fig. 1.8. Algoritmo de razonamiento usando el Método del Valor Verdadero (TVFI)

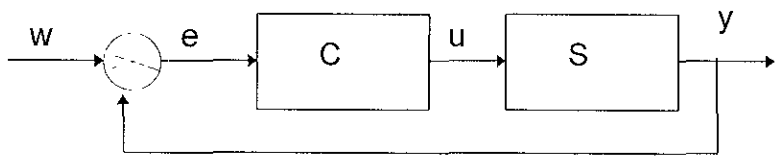


Fig. 1.9. Esquema elemental de control a lazo cerrado.

Relación con los controladores PI, PD y PID convencionales.

Con base en la referencia [12], la relación entre los controladores convencionales y los CLBs es la siguiente:

Controlador PI como un CLB (CLB-PI)

El mecanismo interno de un CLB, describe la relación entre el cambio de la acción de control $\Delta u(k) = u(k) - u(k-1)$ y, el error $e(k)$ y su variación $\Delta e(k) = e(k) - e(k-1)$; donde las entradas y salidas del CLB son las variables de los antecedentes y consecuentes de las reglas contenidas en la base de conocimiento. La ley de control se expresa como:

$$\Delta u(k) = F(K_I e(k), K_p \Delta e(k))$$

donde "F" está en función del proceso de fusificación, base de reglas, proceso de inferencia y defusificación. La expresión anterior es similar a la del controlador Proporcional-Integral, donde la salida es la derivada de la acción de control y, es una función del error y de la derivada del error:

$$\frac{d}{dt}u(t) = K_p \frac{d}{dt}e(t) + K_I e(t)$$

ecuación diferencial

donde $\mathbf{K_P}$ es la constante proporcional y $\mathbf{K_I}$ es la constante integral del controlador PI.

Controlador PD como un CLB (CLB-PD)

Si la variable de salida del CLB no es el cambio de la acción de control $\Delta u(k)$ sino la misma acción de control u(k), se tiene entonces un CLB de tipo PD y la ley de control se expresa como:

$$u(k) = F(k_p e(k), k_D \Delta e(k))$$

Las reglas de este CLB tienen como entradas el error e(k) y el cambio en el error $\Delta e(k)$ y, como salida la acción de control u(k). Este mapeo es similar al conocido

como Proporcional-Derivativo convencional:

$$u(t) = K_p e(t) + K_D \frac{d}{dt} e(t)$$

ecuación diferencial

donde $\mathbf{K_P}$ es la constante proporcional y $\mathbf{K_D}$ es la constante derivativa del controlador PD. La forma de razonamiento (inferencia) del PI y PD son similares, la diferencia entre ellos está en la variable de salida.

Controlador PID como un CLB (CLB-PID)

El CLB-PD puede ser extendido considerando la suma de errores como una variable adicional de entrada. La ley de control se convierte entonces en:

$$u(k) = F(k_p e(k), k_I \Sigma e(k), k_D \Delta e(k))$$

donde $\Sigma e(k) = \sum_{l=0}^{k} e(l-1)$.

Esta función es similar a la ecuación de un controlador PID convencional:

$$u(t) = K_p e(t) + K_I \int e(t)dt + K_D \frac{d}{dt} e(t)$$

ecuación diferencial

donde $\mathbf{K_P}$, $\mathbf{K_D}$ y $\mathbf{K_I}$ son las constantes del controlador PID.

El diagrama básico de un Sistema de Control Lógico Borroso (CLB) tipo PID se muestra en la Fig. 1.10. Si se trata de los tipos PI o PD sólo se tiene que eliminar el diferenciador o el sumador, respectivamente.

Las entradas y salidas de un CLB del tipo PD, PI y, PID son precisamente, las variables de los antecedentes y consecuentes de las reglas contenidas en la base de conocimiento.

El universo del discurso de las variables de entrada y salida, usualmente se definen en el espacio de los números reales. En la práctica, cada universo es restringido a un intervalo que está en función de los posibles valores máximos y mínimos para cada una de las variables. Es decir, se obtiene un intervalo de operación para cada variable involucrada.

Por razones de simplificación y unificación en el diseño de CLBs, así como de su implementación en computadora, es conveniente operar con *universos de discurso normalizados* para las variables de entrada y salida al CLB. En la

mayoría de las aplicaciones se coincide con normalizar las variables al intervalo [-1, 1].

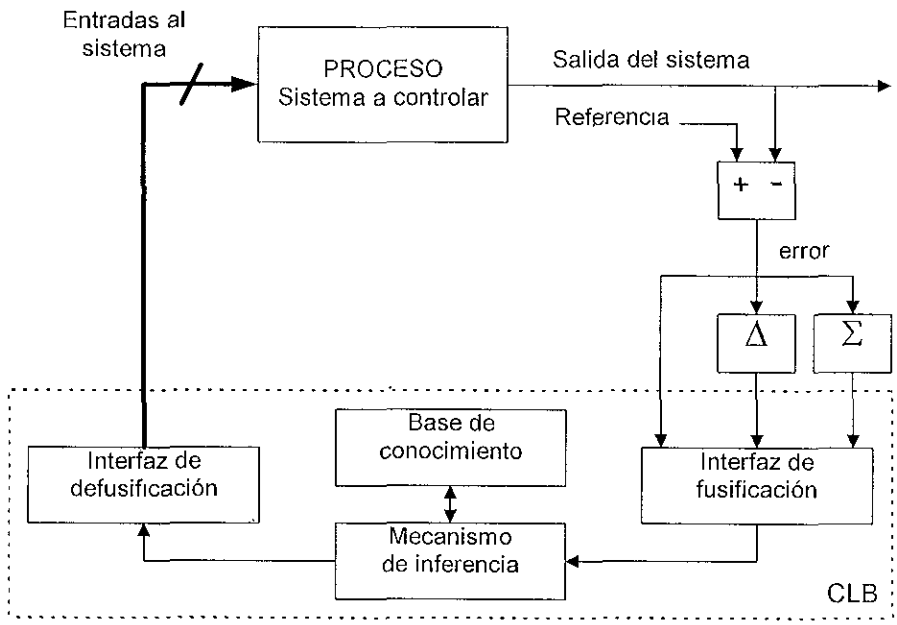


Fig. 1.10. Diagrama básico de un Sistema de Control Lógico Borroso (CLB) tipo PID.

Por ejemplo, la variable del error puede ser normalizada con la expresión:

$$e^*(k)=K_e e(k)$$

donde "e(k)" es el valor real del error, "$e^*(k)$" es la entrada normalizada al controlador y "K_e" un factor de normalización (llamado también factor de escala) que puede ser por ejemplo $K_e=1/e_{max}$. El mecanismo de inferencia se aplicará sobre los valores obtenidos después de la transformación.

De la misma manera que las entradas, el valor obtenido a la salida del controlador puede obtenerse normalizada y, se lleva al universo del discurso mediante una desnormalización con un factor de escala, por ejemplo:

$$u(k)=K_{du}u^*(k)$$

donde "u(k)" es el valor real de la salida , "$u^*(k)$" es la salida normalizada del controlador y "K_{du}" es el factor de desnormalización. Por ejemplo $K_{du}=Salida_{máxima}$.

Las etiquetas de los conjuntos borrosos se definen con palabras en lenguaje natural. Esto se hace con el fin de volver "*más entendibles*" las características que pueden tener las variables y así, tener una visión más sencilla

de lo que puede o debe ser la base de reglas; de tal forma que generalmente se tendrán etiquetas como "*positivo*", "*cero*", "*negativo*", etc.

Tres son los problemas importantes en el diseño de CLBs. El primero es la determinación de las funciones de membresía asociadas con los conjuntos borrosos. El segundo es la construcción de la base de reglas para el CLB y utilización de un sistema de inferencia. El tercero, la sintonización del CLB.

La construcción de la base de reglas es el aspecto más importante y difícil en el diseño de un CLB; de hecho, no existe un método sistemático para obtenerla. En la mayoría de los diseños de controladores borrosos, la base de reglas se obtiene con el conocimiento del sistema a controlar combinado con la experiencia de los operadores que los manejan y, el entendimiento de la teoría de control. También se han diseñado controladores borrosos partiendo de alguna base de reglas estándar (creada para un cierto tipo de problema de control, como la base de reglas de MacVicar-Whelan), a la que se hacen modificaciones específicas para ajustarse a la planta que se desea controlar. Ambas técnicas han producido buenos resultados según la referencia [12].

Diseñado el Controlador Lógico Borroso, el siguiente paso es implementarlo para su prueba y sintonización; pudiendo ser esta última la más complicada, a diferencia de otro tipo de controladores. La sintonización de los CLBs depende de muchos más parámetros y muchas más condiciones, básicamente se efectúa ajustando las constantes de normalización para las variables de entrada y salida. Estos parámetros cambian los intervalos de los universos de discurso y el efecto de éstos en el desempeño del CLB es dramático. Sin embargo, no son los únicos parámetros que se pueden utilizar para sintonizar. En la referencia [14] se sugiere que la sintonización se debe comenzar por los parámetros que tienen un efecto global sobre el CLB y proseguir con aquéllos que produzcan efectos más específicos. Otros parámetros de ajuste pueden ser la modificación de la distribución, forma y simetría de las funciones de membresía, ajuste en las reglas para ejercer acciones de control más drásticas para ciertas condiciones, o viceversa. Es importante remarcar que en el ajuste de un controlador borroso es factible modificar en poca o gran medida cualquiera de los parámetros involucrados; de hecho, algunos ajustes pueden compensar otros que no hayan sido acertados como se muestra en las referencias [12] y [14].

A continuación se hará una revisión de los conceptos de Procesamiento Digital de Imágenes utilizados para el desarrollo del presente trabajo.

1.2 Procesamiento Digital de Imágenes.

En Procesamiento Digital de Imágenes son esencialmente dos áreas de aplicación. La primera, trata de mejorar la información pictórica de la imagen para su mejor interpretación por el sistema de percepción humana. La segunda,

pretende procesar la información de la imagen para lograr un sistema de visión autónomo.

1.2.1 Representación de una imagen digital.

El término *imagen* es utilizado aquí, para referirse a una función de dos variables *f(x, y)* que representa la intensidad de luz; "*x*" y "*y*" denotan coordenadas espaciales y el valor de la función "*f*" en cualquier punto representa el *nivel de gris* de la imagen en ese punto. Por convención, el origen de la imagen es la esquina superior izquierda y tiene coordenadas (0,0). La coordenada "*x*" se refiere al eje de las abscisas y el eje "*y*" al de las ordenadas, en otra palabras "*x*" representa columnas y "*y*" representa renglones.

Una imagen digital es una función *f(x, y)* que ha sido discretizada tanto en coordenadas espaciales como escala de grises. Cada elemento digital recibe el nombre de *pixel.* De esta manera una imagen digitalizada no es más que una matriz de dos dimensiones M×N donde cada elemento representa un cierto valor de tono de gris. El número de tonos de gris por pixel queda determinado por la expresión 2^n donde "*n*" representa el número de bits por pixel. El espacio para el almacenamiento se determina en función de las dimensiones de la imagen y el número de bits por pixel.

1.2.2 Algunas relaciones entre pixeles.

A continuación se muestran algunas relaciones básicas entre pixeles que se utilizan en procesamiento digital de imágenes.

Vecinos de un pixel.

Un pixel *A* con coordenadas (x, y) tiene ocho vecinos como sigue:

- Vecinos verticales y horizontales: (x+1, y), (x-1, y), (x, y+1), (x, y-1).
- Vecinos diagonales: (x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1).

Operaciones aritméticas y lógicas.

Las operaciones aritméticas y lógicas son ampliamente utilizadas en Procesamiento Digital de Imágenes. Las operaciones aritméticas entre los pixeles *A* y *B* se denotan como sigue:

- Adición: *A + B*
- Substracción: *A - B*
- Multiplicación: *A * B*
- División: *A / B*

Las operaciones lógicas se denotan como sigue:

- and : *A and B*
- or: *A or B*
- not: *not A*

Las operaciones anteriores son usadas básicamente en dos formas. La primera, operando pixeles individuales y, la segunda, operando un conjunto de pixeles vecinos. Por ejemplo, la adición de dos imágenes se efectúa haciendo una suma pixel a pixel. El procesamiento por vecinos formula un nuevo término llamado operaciones con *máscara*. La idea de las operaciones con *máscara* es permitir que el valor asignado al pixel procesado esté en función de sí mismo y sus vecinos. Un ejemplo se ilustra en la Fig. 1.11 (a), donde se desea remplazar el valor del pixel *E* por el promedio de los pixeles en una región de 3×3 centrada en el pixel *E*. La operación matemática efectuada es:

$$E=\frac{1}{9}(A+B+C+D+E+F+G+H+I) \tag{1.2}$$

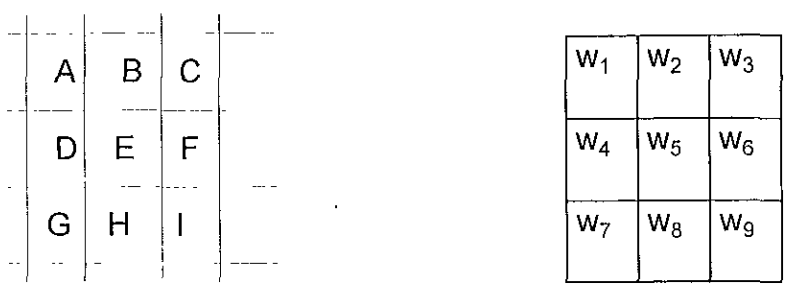


Fig. 1.11 (a) Subárea de una imagen. (b) Máscara de 3×3.

Tomando como referencia la Fig. 1.11 (b), la operación de la ecuación (1.2) puede hacerse centrando la máscara en el pixel *E* y multiplicando cada coeficiente "w_i" por el respectivo pixel de la imagen (A...I) y, sumando estos productos como sigue:

$$E=w_1A + w_2B + w_3C + w_4D + w_5E + w_6F + w_7G + w_8H + w_9I \tag{1.3}$$

si hacemos $w_i=1/9$, i=1,2,...,9, la ecuación (1.3) obtiene el mismo resultado dado por la ecuación (1.2).

El uso de máscaras eligiendo los coeficientes correctos, es útil en Procesamiento Digital de Imágenes; sin embargo, el tiempo de computo es una tarea enorme, dado que la máscara es operada para cada pixel de la imagen.

1.2.3 Forma de procesamiento en el Dominio Espacial.

Muchos de los métodos para el procesamientos de imágenes son efectuados en el dominio espacial. El término *dominio espacial* se refiere a un conjunto de pixeles que conforman una imagen y, *métodos en el dominio espacial* implica procedimientos que operan directamente sobre los pixeles de la imagen. Funciones de procesamiento de imágenes en el dominio espacial pueden ser expresadas como:

$$g(x, y) = T[f(x, y)]$$

donde f(x, y) es la imagen de entrada, g(x, y) la imagen procesada y "T" es un operador sobre "f" definido alrededor de algunos vecinos del pixel (x, y). Se utiliza una sub-imagen (máscara) cuadrada o rectangular centrada en el pixel (x, y) para determinar sus vecinos, como se muestra en la Fig. 1.12. El centro de la máscara se mueve sobre cada pixel de la imagen (comenzando en la esquina superior izquierda) aplicando el operador definido en la máscara a cada localidad (x, y) de "f" para obtener el valor de "g" en (x, y).

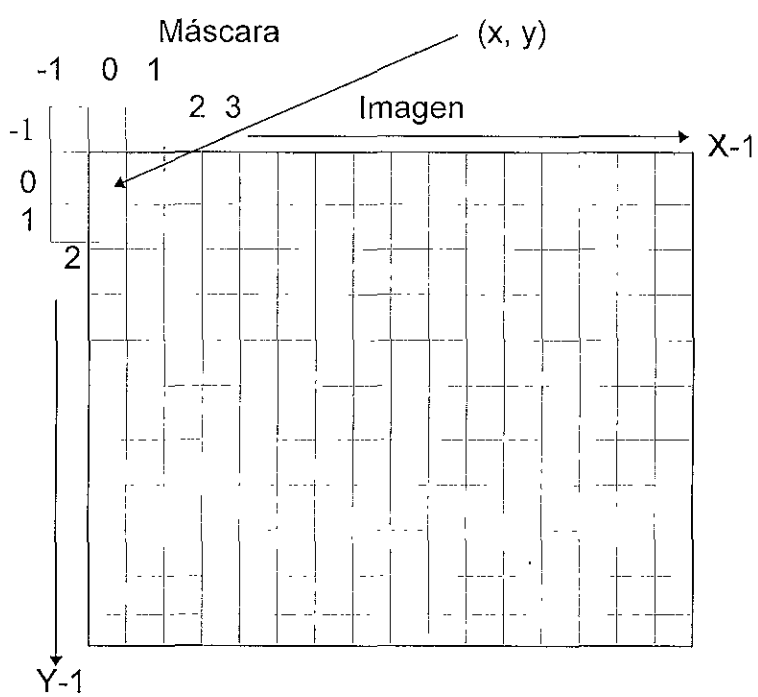


Fig. 1.12. Una máscara de 3×3 sobre el punto (x, y) de una imagen.

La forma más simple de "T" es cuando la vecindad utilizada es de 1×1. En este caso, "g" sólo depende del valor de "f" en el punto (x, y). Esto genera una expresión de la forma *s=T(r)* que es una transformación de niveles de gris, "r" y "s" son variables que denotan el nivel de gris de "f" y "g" en el punto (x, y). Así, la Fig. 1.13 muestra una transformación "T" que utiliza sólo el pixel (x, y) para generar una imagen con mayor contraste, ya que oscurece los pixeles por debajo del nivel "*m*" y los ilumina en caso contrario.

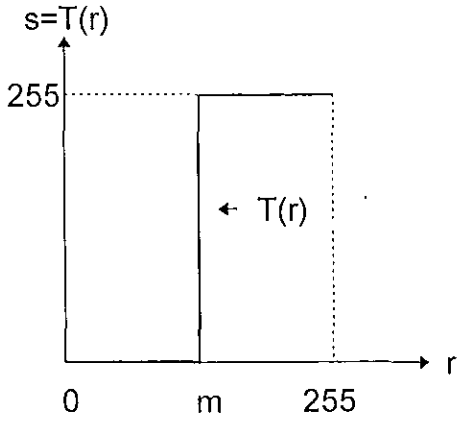


Fig. 1.13. Transformación de niveles de gris.

Una vecindad mayor permite hacer una variedad de procesamientos más sofisticados para una aplicación específica. Sin embargo, el método más general es obtener el valor de "g" con base en una vecindad definida alrededor de f(x, y). Esto se logra con base en el uso de la llamada *máscara* que es un arreglo bidimensional como el mostrado en la Fig. 1.14.

w_1 (x-1,y-1)	w_2 (x,y-1)	w_3 (x+1,y-1)
w_4 (x-1,y)	w_5 (x, y)	w_6 (x+1,y)
w_7 (x-1,y+1)	w_8 (x,y+1)	w_9 (x+1,y+1)

Fig. 1.14. Una máscara genérica de 3×3.

La operación genérica de transformación para la máscara de la Fig. 1.14 se define como:

$$T[f(x, y)]=w_1f(x-1,y-1)+ w_2f(x,y-1)+ w_3f(x+1,y-1)+ w_4f(x-1,y)+ w_5f(x, y)+ w_6f(x+1,y)+ w_7f(x-1,y+1)+ w_8f(x,y+1)+ w_9f(x+1,y+1)$$

1.2.4 Detección de bordes.

Podemos definir un borde como el límite entre dos regiones con distintos niveles de gris. La idea básica para la detección de bordes se basa en el cómputo de una derivada local como se ilustra en la Fig. 1.15. Se muestra, con base en la referencia [15], una imagen de un objeto luminoso sobre un fondo obscuro, el borde a lo largo de una línea horizontal de la imagen y la primera y segunda derivada de ese borde.

La primera derivada de este borde es 0 en todas las regiones con un nivel constante de gris y, asume un valor constante durante una transición de nivel de gris. La segunda derivada es 0 en todas las zonas excepto al inicio y al final de una transición de nivel de gris.

Con base en lo anterior, la magnitud de la primera derivada puede ser utilizada para detectar la presencia de un borde; mientras el signo de la segunda derivada, puede ser utilizado para determinar si un pixel borde está sobre el lado obscuro o luminoso. En la Fig. 1.15 el signo de la segunda derivada es positivo para pixeles sobre el lado obscuro y negativo para aquellos ubicados en el lado luminoso.

1.2.4.1 Filtrado espacial usando el operador Laplaciano.

El *Laplaciano* es un operador que utiliza la segunda derivada y se define:

$$L[f(x, y)] = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial x^2} \qquad (1.4)$$

Dada una función discreta f(x), x=0, 1, 2, ... , la segunda derivada en el punto "x" puede ser aproximada por la expresión:

$$\frac{\partial^2 f(x)}{\partial x^2} \approx f(x+1)-2f(x)+f(x-1)$$

en consecuencia el operador Laplaciano de la ecuación (1.4) puede ser aproximado discretamente como:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial x^2} \approx f(x+1,y)-2f(x,y)+f(x-1,y)+ f(x,y+1)-2f(x,y)+f(x,y-1)$$

$$\approx f(x+1,y)+ f(x-1,y)+ f(x,y+1)+ f(x,y-1)-4 f(x,y)$$

tomando como base la máscara de la Fig. 1.14, el operador *Laplace* se puede aproximar en el punto central (x, y) con nivel de gris w_5 como:

$$L[f(x, y)]=w_2+ w_4+ w_6+ w_8- 4w_5$$

Esta operación responde a transiciones de intensidad como se ilustra en la Fig. 1.15; por consiguiente es un detector de bordes. Puede ser implementada convolucionando la máscara de la Fig. 1.16 con una imagen f(x, y); es decir, se efectúa un procesamiento en el dominio espacial.

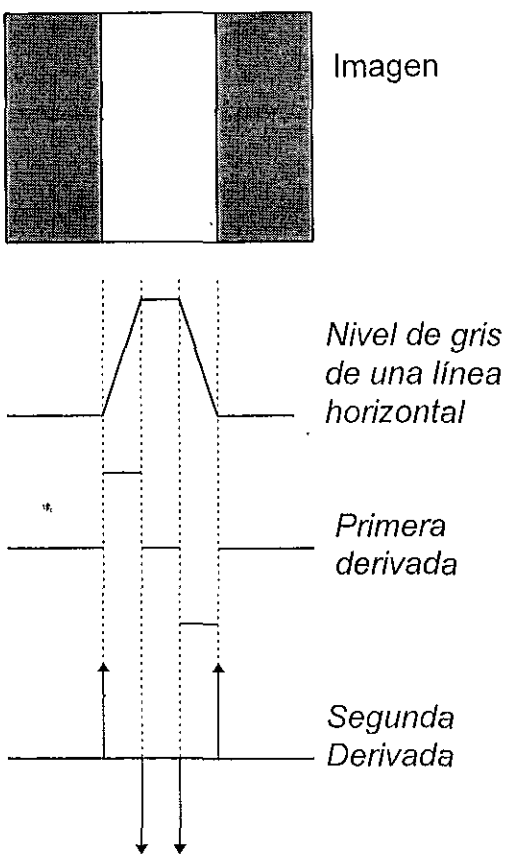


Fig. 1.15. Detección de bordes por el uso del operador derivada.

0	1	0
1	-4	1
0	1	0

Fig. 1.16. Máscara utilizada para calcular el Laplaciano.

1.2.4.2 Filtrado espacial usando Lógica Borrosa.

La versión borrosa para la detección de bordes, con base en la referencia [16], utiliza principalmente un proceso de *fusificación* y uno de *inferencia*. Al igual que la técnica del operador *Laplace*, el procesamiento en el dominio espacial se efectúa con base en una máscara de orden impar; típicamente de 3×3. La máscara utilizada sólo emplea los vecinos verticales y horizontales al pixel (x, y) como se ilustra en la Fig. 1.17.

	w_2 (x,y-1)	
w_4 (x-1,y)	w_5 (x, y)	w_6 (x+1,y)
	w_8 (x,y+1)	

Fig. 1.17. Localidades de la máscara utilizadas en el proceso borroso.

Los tonos de gris de la imagen pertenecen al intervalo [0, 255] y se representan en el intervalo [0, 1] a través de las funciones de membresía llamadas: tonos *bajos (B),* tonos *medios (M)* y tonos *altos (A)*, las cuales proporcionan el grado de pertenencia de cada pixel a tres conjuntos borrosos, como se ilustra en la Fig. 1.18. Las reglas para operar la máscara con la imagen son:

R1: MAX(bajo(x-1,y), medio(x-1,y), alto(x-1,y)) → pertenece[0]= función (B,M,A)
R2: MAX(bajo(x+1,y), medio(x+1,y), alto(x+1,y)) → pertenece[1]= función (B,M,A)
R3: MAX (bajo(x,y-1), medio(x,y-1),alto(x,y-1)) → pertenece[2]= función (B,M,A)
R4: MAX(bajo(x,y+1), medio(x,y+1),alto(x,y+1)) → pertenece[3]= función (B,M,A)
R5: MAX(bajo(x, y), medio(x, y), alto(x, y)) → pertenece[4]= función (B,M,A)

Por cada pixel vecino, incluyendo el pixel central, existe una regla que evalúa según cada función de membresía, el grado de pertenencia del pixel a cada uno de los tres conjuntos borrosos. A partir de estos grados de pertenencia, el operador MAX obtiene el máximo valor de sus argumentos. Como resultado de esta operación se asigna a la variable "*pertenece*" la etiqueta de la función de membresía seleccionada. Después de evaluar las cinco reglas se verifica el contenido de las correspondientes variables "*pertenece*" de la siguiente manera: Si todas las variables "*pertenece*" contienen el mismo número de función de membresía (asignado por la regla) entonces, no existe borde, debido a que los cuatro pixeles vecinos obtienen su mayor grado de pertenencia al mismo conjunto

borroso. Si alguna de las variables "*pertenece*" no contiene el mismo número de función de membresía entonces, existe un borde en el pixel central.

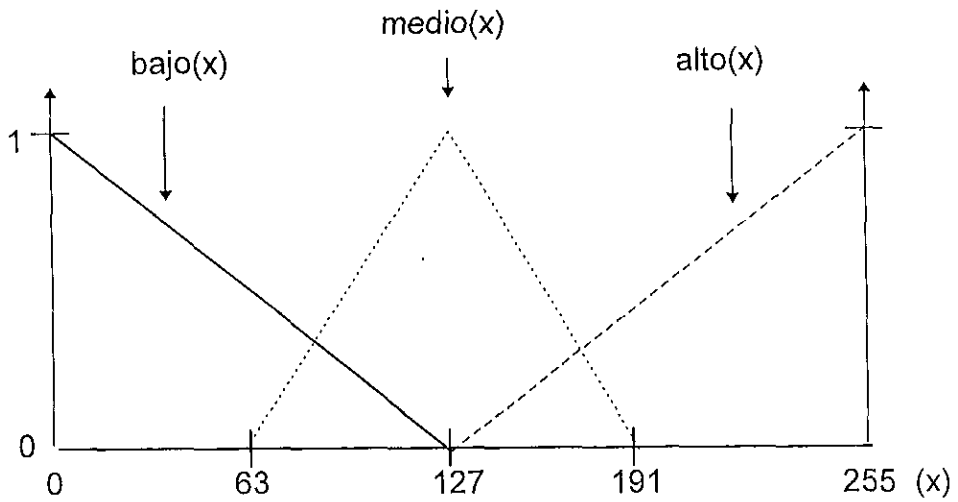


Fig. 1.18. Funciones de membresía.

1.2.4.3 Filtrado espacial usando Comparación de Mínima Distancia.

El procesamiento utilizando Comparación de Mínima Distancia es bastante simple; se efectúa con una máscara igual a la de la Fig. 1.17. La parte esencial del procesamiento es fijar un *radio* de comparación consistente de un valor numérico entre 0 y 255. Este *radio* sirve para decidir que tan pronunciada se desea la detección de bordes en la imagen y la susceptibilidad a detección de ruido. El procedimiento consiste en obtener la diferencia en valor absoluto entre el pixel central de la máscara y cada uno de sus cuatro vecinos como sigue:

- diferencia_1=absoluto(máscara(x,y)-máscara(x,y-1))
- diferencia_2=absoluto(máscara(x,y)-máscara(x,y+1))
- diferencia_3=absoluto(máscara(x,y)-máscara(x-1,y))
- diferencia_4=absoluto(máscara(x,y)-máscara(x+1,y))

La comparación de cada diferencia con "*radio*" permite decidir de la siguiente manera si existe un borde en el pixel central:

- si diferencia_1>*radio* ó diferencia_2>*radio* entonces existe borde vertical.
- si diferencia_3>*radio* ó diferencia_4>*radio* entonces existe borde horizontal.

El valor de *radio* tiene un significado especial, dado que entre menor sea este, más susceptible es la detección de bordes y de ruido.

CAPÍTULO 2
ALGORITMOS PARA LA DETECCIÓN, SEGUIMIENTO Y SIMULACIÓN DEL OBJETO EN MOVIMIENTO.

Este capítulo muestra los diseños y resultados de los algoritmos que permiten hacer la detección y seguimiento del objeto en movimiento, así como el algoritmo utilizado para simular el sistema físico, que involucra la conjunción de los algoritmos anteriores.

2.1 Simulador del Sistema Físico.

2.1.1 Objetivo.

El sistema físico para el seguimiento de un objeto en movimiento está constituido por un sistema de adquisición de datos, un sistema de procesamiento de la imagen y seguimiento del objeto y, un sistema para el control del movimiento de la cámara. La creación de un simulador permite en primer lugar, evaluar el desempeño de los algoritmos de procesamiento y seguimiento del objeto, que constituyen la parte más importante del sistema físico. En segundo lugar, conjunta estos algoritmos con una simulación de los dos subsistemas restantes en un ambiente muy parecido al real.

El simulador tiene por objetivos:

- Generar una trayectoria de desplazamiento para el objeto.

- Simular el movimiento del objeto sobre la trayectoria generada.
- Simular la cámara de video y su movimiento.
- Simular la adquisición periódica de imágenes por la cámara de video.
- Evaluar el desempeño de los algoritmos de procesamiento para detección del movimiento del objeto.
- Evaluar el desempeño de los algoritmos para el seguimiento del objeto.

Los algoritmos de procesamiento para la detección del movimiento del objeto se basan en las técnicas de reconocimiento de bordes descritas en el Capitulo 1.

2.1.2 Estructura.

El simulador está desarrollado por módulos en C++, de manera que los algoritmos de procesamiento de imágenes y seguimiento del objeto pueden ser intercambiados fácilmente por otros que tengan diseños diferentes; de esta manera es posible analizar diversas técnicas sin necesidad de modificar gran parte del simulador. A continuación se describe brevemente el funcionamiento de cada módulo:

a) Trayectoria de desplazamiento para el objeto.

Con el objetivo de comparar el desempeño de las técnicas utilizadas, este módulo es capaz de generar trayectorias fijas para el movimiento del objeto. También puede generar trayectorias aleatorias definidas por el usuario mediante el uso del ratón. En síntesis puede generar las siguientes trayectorias, las cuales se ilustran en la Fig. 2.1:

- Horizontal con distribución constante con incremento unitario de pixeles.
- Horizontal con distribución constante con incremento de cuatro pixeles.
- Horizontal con distribución no constante con incremento lineal creciente.
- Horizontal con distribución no constante con incremento lineal decreciente.
- Aleatoria generada por el usuario mediante el uso del ratón.

b) Simulación del movimiento del objeto sobre la trayectoria generada.

El objeto utilizado por el simulador está constituido por una imagen con dimensiones de 128×128 pixeles. El tamaño se elige así por el tiempo de procesamiento y, limitaciones con respecto a la máxima resolución del monitor; dado que imágenes con dimensiones mayores no dispondrían de una zona amplia para simular el movimiento. Esta simulación consiste en posicionar consecutivamente la esquina superior izquierda del objeto sobre cada punto de la trayectoria. Un parámetro denominado como *periodo de movimiento del objeto* y representado como *T_objeto*, permite controlar la velocidad del objeto sobre la trayectoria. El parámetro puede ser establecido por el usuario, de forma tal que el

desplazamiento entre cada posición de la trayectoria sea cada *T_objeto* centésimas de segundo.

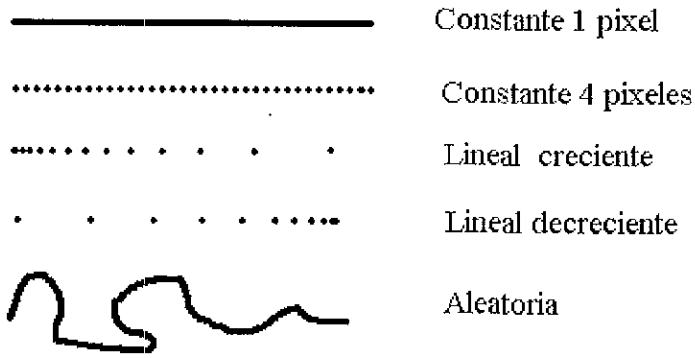


Fig. 2.1. Diversas trayectorias generadas por el simulador.

La simulación del desplazamiento del objeto implica, en primer lugar, borrar la imagen de la posición actual y, en segundo lugar, copiarla en la nueva posición. No obstante, este procedimiento trae dos problemas:

- Incrementa el tiempo consumido para el despliegue, debido a que involucra un borrado y un copiado de la imagen completa.
- Produce un efecto de parpadeo, producto de borrar y después copiar.

Ambos problemas son resueltos eliminando solamente la sección de la imagen que no será sobrescrita. El tiempo se reduce a un despliegue más el borrado de la sección que debe ser eliminada. Así el parpadeo desaparece porque las partes sobrescritas no son borradas. La Fig. 2.2 ilustra las cuatro posibilidades en función del movimiento del objeto. Para cada caso la zona de intersección entre ambos cuadros será sobrescrita, el resto será borrada. El cuadro continuo representa la posición actual de la imagen (X_O, Y_O) y, el cuadro discontinuo representa la nueva posición (X_d, Y_d) ambas con referencia a su esquina superior izquierda en el orden columna, renglón respectivamente. Las opciones de la Fig. 2.2 son identificadas por el simulador usando las condiciones que aparecen bajo cada caso.

c) Simulación de la cámara de video y su movimiento.

La cámara de video simulada tiene un tamaño de 128×128 pixeles y una referencia *(pos_act_col, pos_act_ren)* que indica su posición en relación a la zona de despliegue del simulador. Estas coordenadas son utilizadas, en primer lugar, para hacer la adquisición de imágenes por la cámara y, en segundo lugar, por el algoritmo para el seguimiento del objeto. Con una finalidad solamente visual, el

simulador despliega la imagen adquirida por la cámara y la actualiza cada *T_camara* centésimas de segundo.

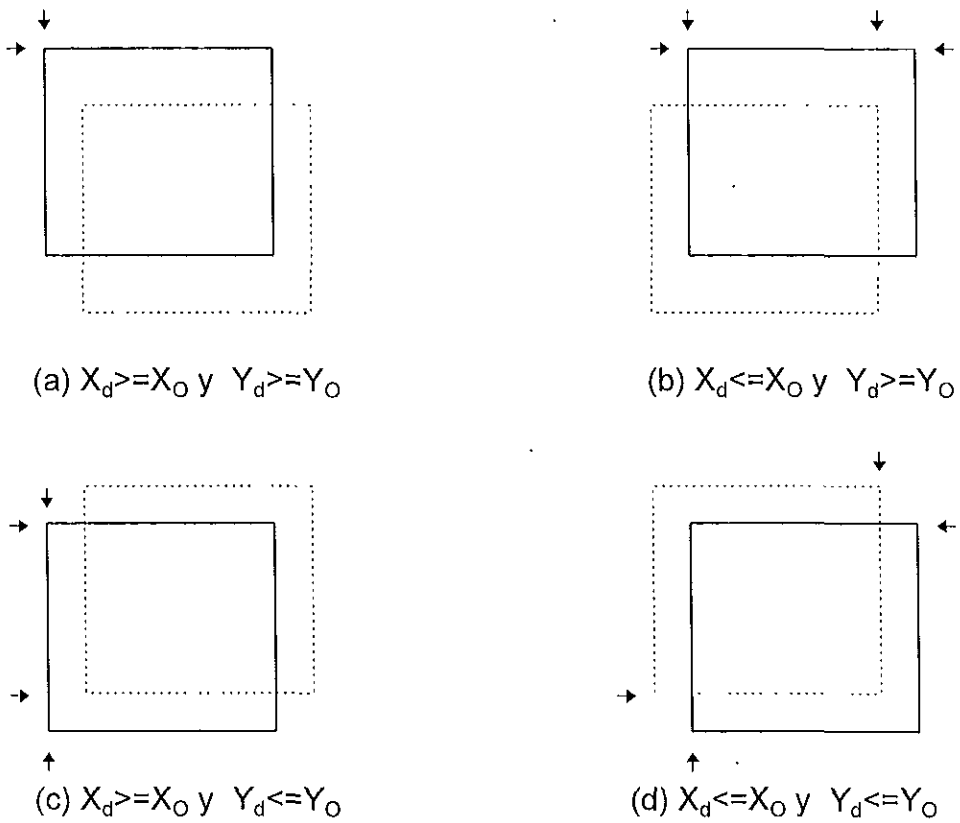


Fig. 2.2. Cuatro casos considerados en la simulación del desplazamiento del objeto.

d) Simulación de la adquisición periódica de imágenes por la cámara de video.

La cámara de video está ubicada en la posición *(pos_act_col, pos_act_ren)* en relación a la zona de simulación. Las adquisiciones se simulan copiando el contenido del cuadro definido por las coordenadas *(pos_act_col, pos_act_ren)* y *(pos_act_col+128, pos_act_ren+128)* en el momento que sea solicitada por la rutina de procesamiento del simulador. El intervalo de tiempo entre dos adquisiciones consecutivas está en función directa del tiempo de procesamiento por imagen; en consecuencia, es un indicador de la velocidad del sistema global.

2.1.3 Diagrama de flujo.

El diagrama de flujo del simulador se ilustra en la Fig. 2.3. Cada número ubicado a la derecha o izquierda es usado para hacer una breve descripción de las acciones hechas por ese bloque, como sigue:

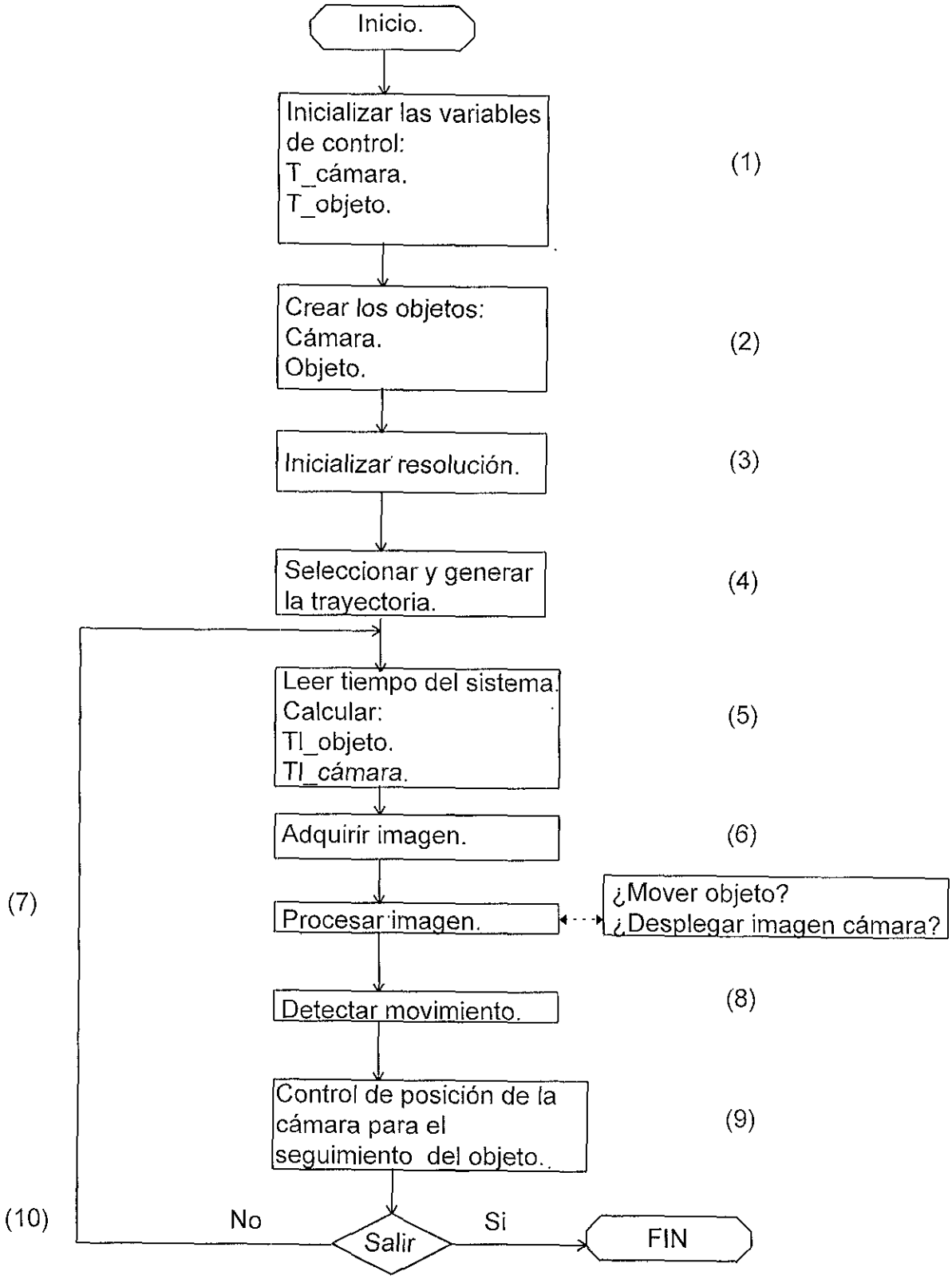


Fig. 2.3. Diagrama de flujo del simulador.

(1) Inicializa las variables de control *T_objeto* y *T_cámara* que indican el periodo para el movimiento del objeto sobre la trayectoria, así como el periodo para el despliegue del campo de visión de la cámara. Estas variables pueden ser modificadas en cualquier momento por el usuario.

(2) Crea los espacios de memoria necesarios para almacenar el objeto en movimiento y la imagen capturada por la cámara de video. También crea variables adicionales usadas internamente para el manejo del objeto y la cámara, como son: paleta de colores, tamaño de la imagen, ubicación de la zona para el despliegue de las imágenes capturadas, etc.

(3) Inicia la resolución en la cual trabajará el simulador. Se tienen disponibles las siguientes resoluciones todas ellas en 256 tonos de gris: 1024×768, 800×600 y 640×480. El incremento en la resolución aumenta la zona disponible para hacer la simulación. Para cada una de las resoluciones se inicializa la paleta de colores, así como las posiciones para los despliegues efectuados. Después de inicializar la resolución se despliega el simulador, con los parámetros establecidos por omisión.

(4) En este punto se selecciona la trayectoria a utilizar según los tipos mostrados en la Fig. 2.1, siendo la trayectoria aleatoria la seleccionada por omisión, de manera que el simulador espera a que el usuario trace la trayectoria haciendo uso del ratón.

(5) Se lee el tiempo actual del sistema que es utilizado para calcular dos tiempos muy importantes. El primero llamado *Tl_objeto* que indica cuando el objeto debe ser desplazado una posición más sobre la trayectoria, no importando que en ese instante se este efectuando procesamiento sobre una imagen adquirida. Por lo tanto, *Tl_objeto* debe ser monitoreado continuamente por el simulador. Este tiempo se calcula:

$$Tl\_objeto = Tiempo\ actual\ del\ sistema + T\_objeto$$

como se observa, el objeto se desplaza cada *T_objeto* centésimas. El segundo tiempo se llama *Tl_cámara,* también monitoreado por el simulador de la misma manera que *Tl_objeto*, indica cuando se debe actualizar la zona de despliegue de la cámara con la imagen más reciente que se tenga. Este tiempo se calcula:

$$Tl\_cámara = Tiempo\ actual\ del\ sistema + T\_cámara$$

de esta forma, la zona de despliegue del campo de visión de la cámara estará modificándose cada *T_cámara* centésimas.

(6) La siguiente imagen a utilizar para el procesamiento se captura usando la rutina de simulación de la cámara y sus coordenadas actuales.

(7) Esta es la parte más densa en procesamiento iterativo, dado que se encarga de extraer las características de la imagen que posteriormente van a permitir detectar si el objeto se movió. Por consiguiente, es aquí donde el simulador pasa la mayor parte del tiempo y en consecuencia se inserta código que le permite monitorear continuamente las variables *Tl_objeto y Tl_cámara*; las cuales comparándolas con el tiempo actual del sistema, permiten saber si se debe interrumpir el procesamiento para desplazar el objeto a la siguiente posición y/o actualizar la zona de despliegue del campo de visión de la cámara.

(8) Utiliza los resultados del procesamiento iterativo sobre la imagen, para determinar una característica final que represente la posición del objeto y poder detectar el movimiento.

(9) Utiliza las posiciones del objeto determinadas en el paso anterior, para poder controlar el posicionamiento de la cámara usando las coordenadas *(pos_act_col, pos_act_ren).*

(10) Interrumpe el proceso cuando el usuario desea terminar la ejecución del simulador.

2.1.4 Ilustración del simulador.

La Fig. 2.4 ilustra el simulador completo; el cual se encuentra dividido en cuatro zonas principales:

- La zona superior izquierda representa el menú, donde el usuario puede cambiar los periodos de las variables *T_objeto* y *T_cámara*, el objeto a utilizar para la simulación y la trayectoria a definir para el desplazamiento del objeto.

- La zona central superior presenta los valores de las variables *T_objeto* y *T_cámara*, así como los mensajes generados por el simulador.

- La zona superior derecha representa la zona de despliegue del campo de visión de la cámara.

- La zona inferior representa el área de simulación, donde se ilustra la trayectoria y el objeto a seguir. La posición de la cámara no se visualiza, pero las imágenes observadas se muestran en la esquina superior derecha.

2.2 Extracción de la posición del objeto.

Determinando la posición del objeto en relación al campo de visión de la cámara CCD, en el caso del simulador de 128×128 pixeles, es posible detectar movimiento mediante la comparación de dos posiciones consecutivas. Esta posición se obtiene efectuando un procesamiento digital de extracción de bordes

sobre la imagen adquirida, los cuales representan la información más relevante que sirve para definir la posición del objeto. Posteriormente estos bordes son utilizados para calcular un centroide que finalmente representa la posición del objeto.

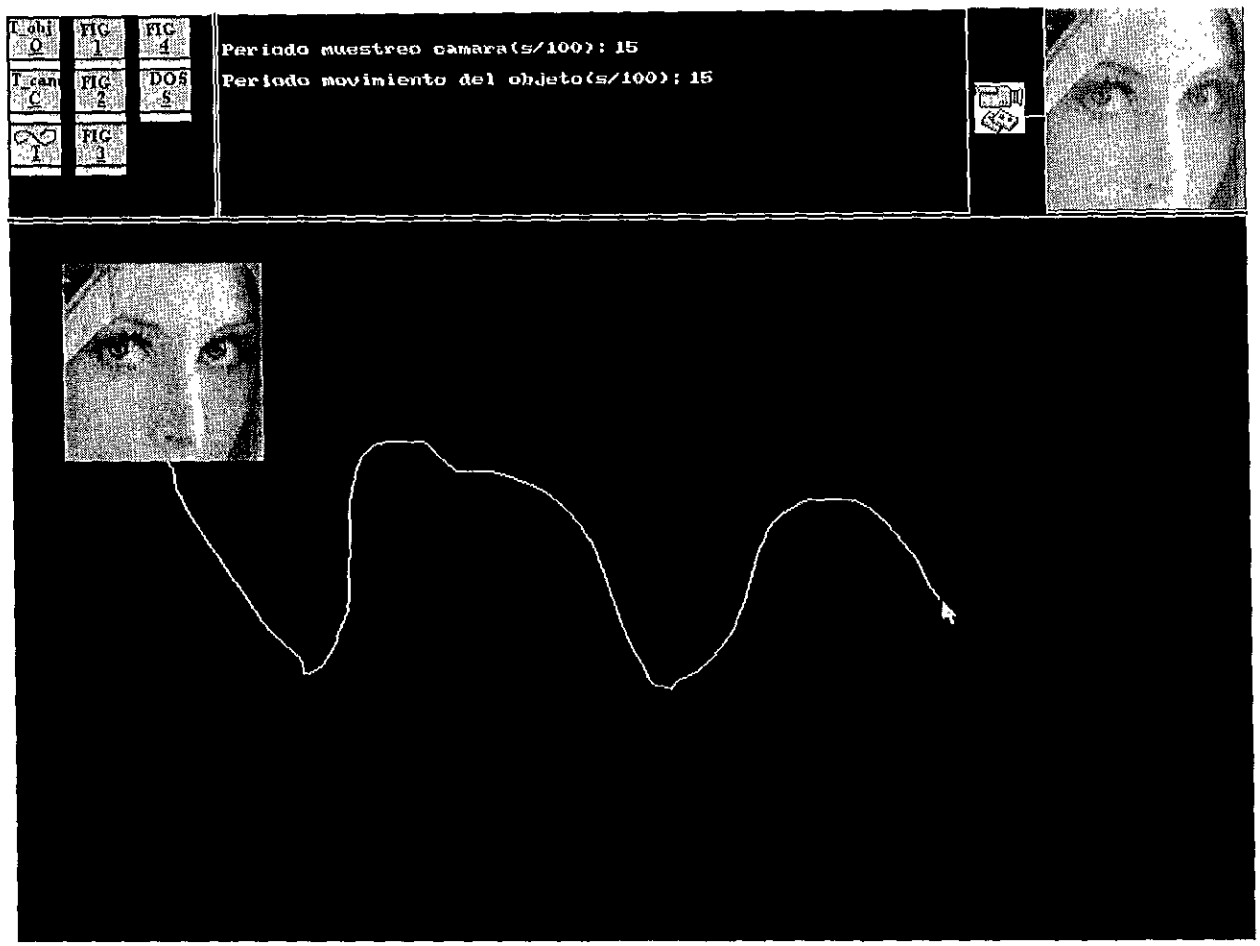


Fig. 2.4. Simulador del sistema físico.

La extracción de bordes es efectuada con base en las tres técnicas analizadas en el Capitulo 1. Estas son evaluadas usando el simulador considerando los siguientes dos parámetros cruciales para el sistema físico:

- Velocidad de procesamiento.
- Susceptibilidad a la detección de ruido.

La primera, acerca o aleja nuestro sistema de un procesamiento de 30 cuadros por segundo (velocidad de adquisición en el estándar NTSC); mientras que la segunda afecta la exactitud en el cálculo del centroide, evitando detectar bordes falsos como parte del objeto.

2.2.1 Procesamiento.

Independientemente de la técnica utilizada para la detección de bordes, el procesamiento se efectúa espacialmente por cada pixel de la imagen adquirida, haciendo uso de los pixeles vecinos al pixel procesado usando una máscara igual a la explicada en el Capítulo 1.

El código de cada técnica y el diccionario de datos está incluido en el listado del simulador en el Apéndice A. Para cada una de las técnicas, la imagen resultante representa sólo sus bordes, no su intensidad; es decir, dónde existe borde se representa con el máximo tono de gris (255) y donde no hay borde con el mínimo tono de gris (0). Para todos los casos las imágenes fuentes son de tamaño 128×128 y 256 tonos de gris.

2.2.1.1 Resultados del filtrado usando el filtro Laplaciano.

La imagen de bordes resultante se ilustra en la Fig. 2.5. Se observa que el filtro *Laplaciano* es muy sensible a la detección de bordes y en consecuencia, a ruido incorporado por factores ambientales e intrínsecos del sistema de adquisición.

El desempeño en velocidad de procesamiento se muestra en la Tabla 1, donde se observa que en un procesador Pentium a 90 Mhz es posible procesar aproximadamente 2.6 imágenes por segundo. Esto dista mucho de acercarse al estándar NTSC (adquisición de 30 cuadros por segundo).

a) Imagen original.

b) Imagen procesada.

Fig. 2.5. Imagen original y de bordes usando el filtro Laplaciano.

Tabla 1

CPU	486 /33	486/66	586/90
Tiempo ms	2690	990	380

Tabla 1. Velocidad de procesamiento para el filtro Laplaciano.

2.2.1.2 Resultados del filtrado usando Lógica Borrosa.

La imagen de bordes resultante se ilustra en la Fig. 2.6. Como se puede ver, la técnica borrosa es menos sensible a la detección de bordes y en consecuencia, a ruido incorporado por factores ambientales e intrínsecos al sistema de adquisición.

La Tabla 2 muestra el desempeño de esta técnica en un procesador Pentium a 90 Mhz, que dista mucho de acercarse al estándar NTSC, siendo muy semejante a la técnica usando el filtro *Laplaciano*.

a) Imagen original. b) Imagen procesada.

Fig. 2.6. Imagen original y de bordes usando Lógica Borrosa.

Tabla 2

CPU	486 /33	486/66	586/90
Tiempo ms	2690	980	390

Tabla 2. Velocidad de procesamiento para la técnica de Lógica Borrosa.

2.2.1.3 Resultados del filtrado usando Comparación de Mínima Distancia.

Las imágenes de bordes resultantes para diversos valores del parámetro "*radio*" de comparación se ilustran en la Fig. 2.7. Se observa que esta técnica es sensible a la detección de bordes y en consecuencia al ruido, en función del valor utilizado para el parámetro "*radio*". Si éste es pequeño, la detección de bordes es mejor; sin embargo, aumenta su susceptibilidad al ruido.

El desempeño en velocidad de procesamiento se muestra en la Tabla 3, se observa que en un procesador Pentium a 90 Mhz es posible procesar 9 imágenes por segundo. Este procesamiento es más rápido que el obtenido con las técnicas anteriores, adaptándose mejor a los requerimientos del sistema físico.

a) Imagen original b) Imagen con radio=30 c)Imagen con radio=20

d) Imagen con radio=10 e) Imagen con radio=5

Fig. 2.7. Imagen original y de bordes usando Comparación de Mínima Distancia.

Tabla 3

CPU	486 /33	486/66	586/90
Tiempo ms	710	280	110

Tabla 3. Velocidad de procesamiento para la técnica de Comparación de Mínima Distancia.

2.2.2 Cálculo del centroide.

Finalmente, la posición del objeto se localiza obteniendo el centroide de la imagen de bordes encontrada. Este se extrae calculando un promedio estadístico para cada una de las coordenadas con base en las siguientes expresiones:

$$C_X = \frac{1}{N}\sum_{i=1}^{N} X_i \qquad C_Y = \frac{1}{N}\sum_{i=1}^{N} Y_i$$

donde (C_X, C_Y) son las coordenadas del centroide, N es el número de pixeles borde encontrados y (X_i,Y_i) son las coordenadas de los pixeles bordes.

2.3 Sistemas para el seguimiento del objeto.

Las posiciones obtenidas después del procesamiento de la imagen son utilizadas por un sistema encargado de hacer el seguimiento del objeto, generando las acciones de control necesarias para evitar que salga del campo de visión de la cámara. En esta sección se presentan dos sistemas; el primero, hace una Corrección Simple utilizando el error; el segundo, efectúa una corrección utilizando un Controlador Lógico Borroso (CLB).

Los resultados obtenidos en cada uno de los sistemas anteriores serán mostrados en una serie de gráficas, las cuales ilustran el desempeño para cada una de las trayectorias definidas en la Fig. 2.1. Para interpretarlas es necesario considerar los siguientes puntos salvo se indique lo contrario:

- Se ilustran cuatro trayectorias bien definidas, con incrementos: Constante de un pixel, de cuatro pixeles, lineal creciente y lineal decreciente.
- Se ilustra un periodo y medio de cada trayectoria.
- Se analiza sólo la coordenada X de las trayectorias.
- El eje de las abscisas ilustra el número de muestras tomadas, correspondientes al número de centroides calculados durante el proceso de ejecución.
- El eje de las ordenadas ilustra el valor de la coordenada X del centroide calculado en esa muestra.
- Un valor de 130 pixeles en el eje de las ordenadas indica que el objeto ha salido del campo de visión de la cámara.

2.3.1 Control usando Corrección Simple del Error.

El seguimiento del objeto se logra fijando una referencia que se compara con el centroide del objeto, generando un error que se utiliza para calcular las correcciones a efectuar sobre las posiciones de la cámara, como sigue:

X_W=64
e=X_W - X_O
pos_act_col=*pos_act_col* - e

donde X_W es la referencia (centro de visión de la cámara), "e" es el error, X_O la coordenada X del centroide del objeto y *pos_act_col* es la coordenada X de la cámara. Se muestra sólo el ajuste para la coordenada X, ya que el mecanismo es idéntico para la coordenada Y. De esta manera si el centroide X_O sobrepasa la referencia, indica que el objeto está saliendo del campo de visión de la cámara por el extremo derecho, generando así un error negativo que al restarse a la posición de la cámara, genera la corrección. El desempeño de este sistema se ilustra en la Fig. 2.8 para cuatro trayectorias diferentes y *T_objeto=25 centésimas*. El objeto es perdido para las trayectorias lineal creciente y decreciente. Para las dos restantes esto no ocurre; sin embargo, para todas las trayectorias la respuesta está ubicada fuera de la referencia.

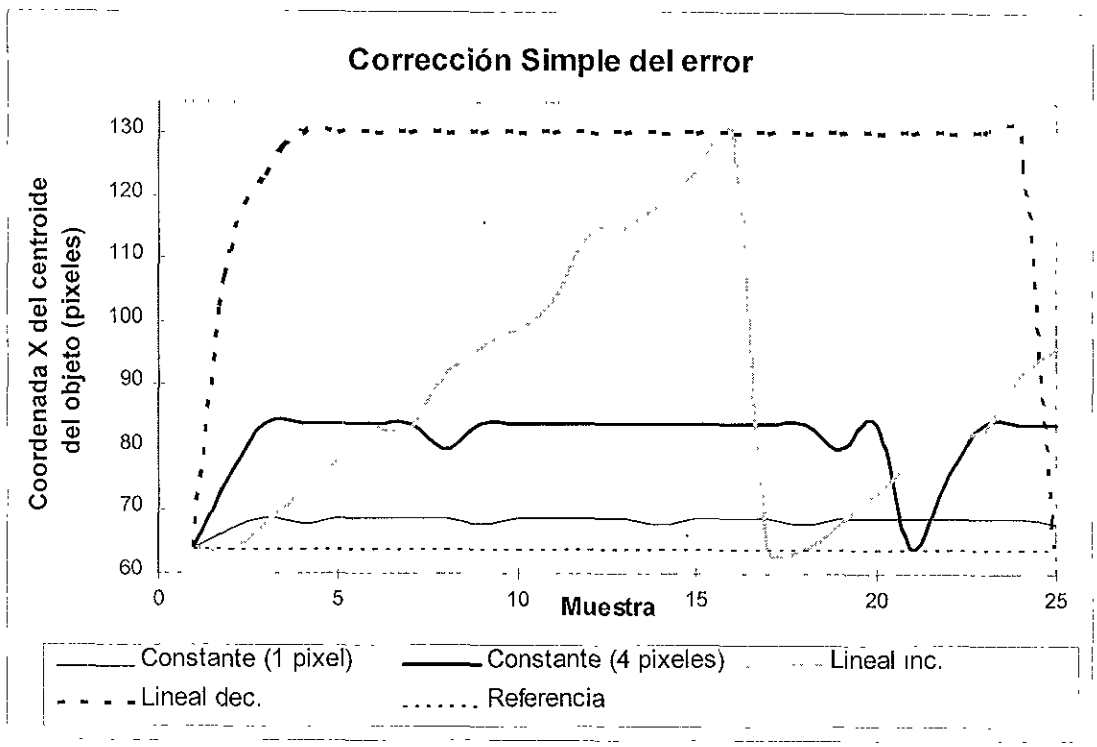


Fig. 2.8. Seguimiento del objeto utilizando corrección simple del error.

2.3.2 Control Lógico Borroso.

2.3.2.1 Objetivo.

El objetivo del Sistema de Control Lógico Borroso (CLB), consiste en mantener el centroide del objeto en movimiento en el centro de visión de la cámara; así como incorporar una predicción en el cálculo de las coordenadas de ajuste para

adelantarse al movimiento del objeto. La Fig. 2.9 ilustra este objetivo. El cuadro mayor representa el área de visión de la cámara, $W(X_W, Y_W)$ representa su centro de visión y el punto $O(X_O,Y_O)$ el centroide del objeto referenciado a la posición de la cámara.

La esquina superior izquierda de la cámara corresponde a la coordenada (0,0), y, la esquina inferior derecha a la coordenada (127,127). Se desea que el punto "O" siempre coincida con el punto "W". El CLB calculará la acción de control necesaria para llevar el centro de visión de la cámara hacia la posición del centroide del objeto, estas acciones de control son por cada una de las coordenadas que definen la posición de la cámara. El sistema de control utilizado se muestra en la Fig. 2.10, es un controlador PI (proporcional - integral) como un CLB y se denota como CLB-PI. Se ilustra el diagrama a bloques para una sola coordenada.

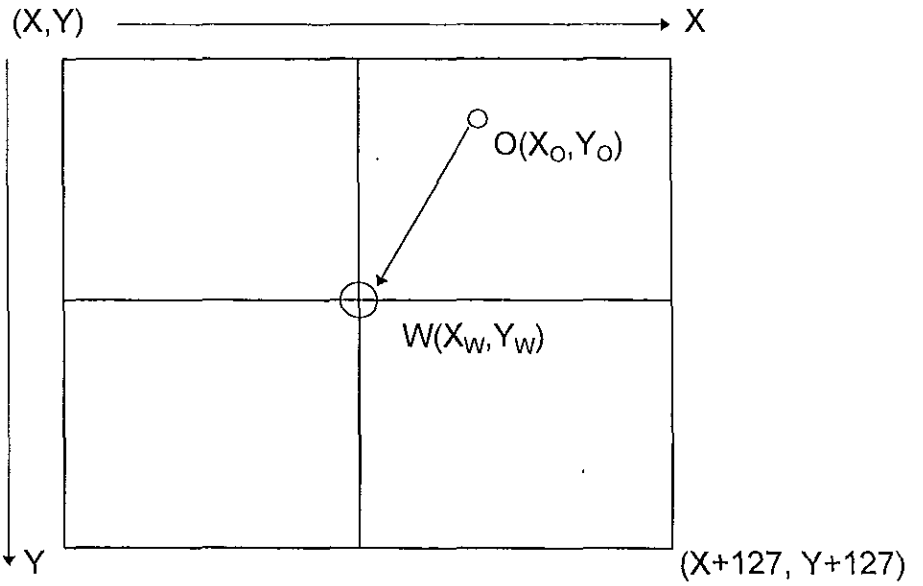


Fig. 2.9. Objetivo del Controlador Lógico Borroso.

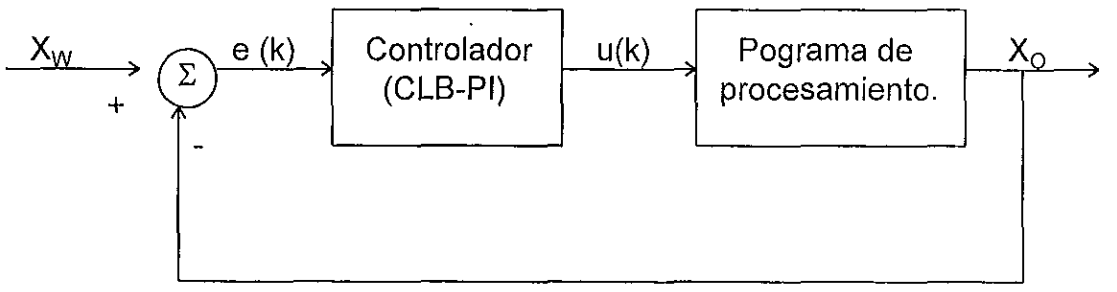


Fig. 2.10. Diagrama a bloques del Control Lógico Borroso para el posicionamiento de la cámara.

X_W es la referencia deseada para la coordenada X_O del centroide del objeto, e(k) representa el error que es la entrada al CLB, éste entrega la acción de control que consiste en un valor para la coordenada correspondiente a la posición de la cámara. El programa de procesamiento ejecuta la acción de control, así como el cálculo del centroide de las imágenes adquiridas.

El error es calculado como:

$$e(k)=X_W(k)-X_O(k) \tag{2.1}$$

donde "k" se refiere al estado actual.

La Fig. 2.11 ilustra los componentes internos del CLB-PI:

- Error, *calculado como se muestra en la ecuación* (2.1)
- Variación del error, calculado como:

$$e(k)= e(k)-e(k-1) \tag{2.2}$$

- Incremento en la acción de control $\Delta u(k)$.
- Estructura interna del CLB constituida por la interfaz de fusificación, mecanismo de inferencia, base de conocimiento e interfaz de defusificación.
- Acción de control calculada como $u(k)=u(k-1)+ \Delta u(k)$

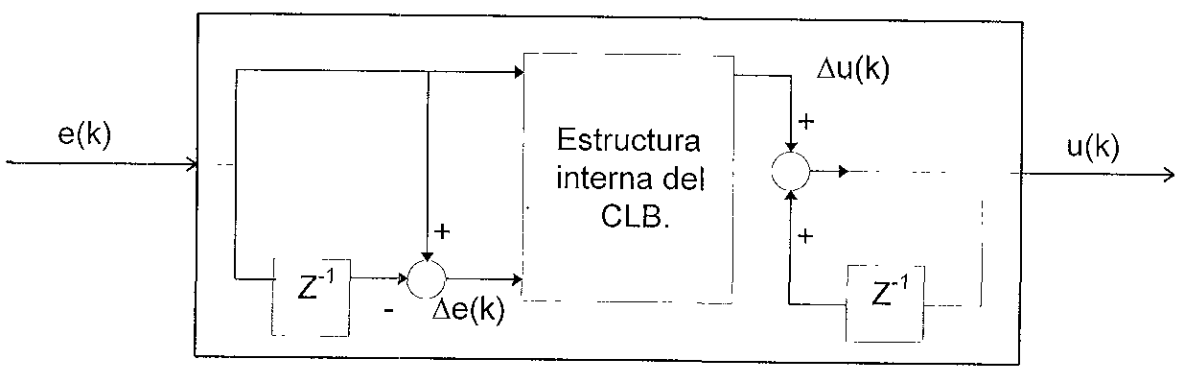


Fig. 2.11. Componentes internos del CLB-PI.

2.3.2.2 Diseño del CLB.

a)Variables de entrada y salida.

Las variables de entrada internas al CLB-PI son el error y su variación. La variable de salida interna $\Delta u(k)$ es un incremento algebraico de la acción de control, que es la posición de la cámara como se ilustra en las Figs. 2.10 y 2.11.

b) Definición de parámetros del CLB-PI.

Se utiliza una imagen de 128×128 pixeles, con base en ella se calculan los intervalos de las variables de entrada y salida, así como los valores deseados para la referencia como sigue:

- Las referencias para ambas coordenadas son $X_W=Y_W=64$ que corresponden exactamente al centro de la cámara, lugar donde se desea mantener el objeto seguido.

- El intervalo del error está definido por sus extremos máximos y mínimos que se definen con base en la referencia y, el valor máximo y mínimo de las coordenadas del centroide:

  $X_{Omax}=128$; $X_{Omin}=0$
  $e_{max}=X_W-X_{Omin}=64$; $e_{min}=X_W-X_{Omax}=-64$
  Por tanto, el intervalo del error "e" es [-64, 64].

- El intervalo para la variación del error está determinado por los extremos máximo y mínimo del error como sigue:

  $\Delta e_{max}=e_{max}-e_{min}=128$; $\Delta e_{min}=e_{min}-e_{max}=-128$
  Por tanto, el intervalo de la variación del error "Δe" es [-128, 128].

- El intervalo de operación de la variable de salida debe ser capaz de corregir un caso extremo. Este tiene lugar, cuando el objeto pasa del campo de visión izquierdo al derecho de la cámara, siendo necesaria una corrección igual al ancho de la imagen capturada, en consecuencia el intervalo de la variable de salida "Δu(k)" es [-128, 128].

c) Normalización de los intervalos.

Cada intervalo para las variables de entrada y salida son normalizados entre [-1 1]. Los parámetros de normalización permiten posteriormente la sintonización del CLB-PI

- El parámetro de normalización se denota por K_e con valor 1/64, el error normalizado se obtiene como:

$$e^*(k)=K_e e(k)$$

- El parámetro de normalización se denota por K_d con valor 1/128, la variación del error normalizado se obtiene como:

$$\Delta e^{*}(k)=K_q\,\Delta e(k)$$

- El parámetro de desnormalización se denota por K_{du} con valor de 128, la salida de control desnormalizada se calcula como:

$$u(k)= K_{du}\,\Delta u^{*}(k)$$

d) Funciones de membresía y matriz de reglas.

En la Fig. 2.12 se muestran las tres funciones de membresía utilizadas por las variables del CLB-PI, la diferencia entre ellas radica en el parámetro de normalización utilizado para cada variable. Estas funciones de membresía permiten evaluar el error, su variación, el incremento en la acción de control y obtener su grado de pertenencia a tres conjuntos: *Negativo (N), Zero (Z), y Positivo (P).*

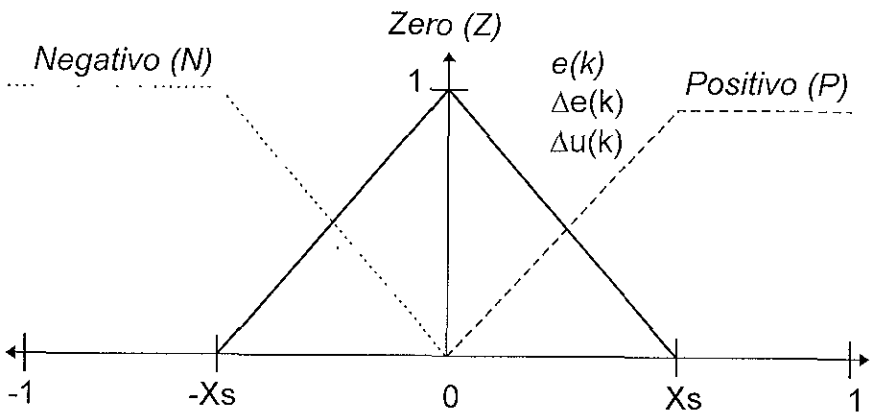


Fig. 2.12. Funciones de membresía para e(k), Δe(k), y Δu(k) con K_e=1/64, K_d=1/128 y K_{du} = 128.

Las ecuaciones que definen las funciones de membresía de la Fig. 2.12 son:

$$N=\begin{cases} 1 & ;si\ x <= -Xs \\ (-1/Xs)x & ;si\ -Xs < x <= 0 \\ 0 & ;si\ x > 0 \end{cases} \qquad P=\begin{cases} 0 & ;si\ x < 0 \\ (1/Xs)x & ;si\ 0 <= x < Xs \\ 1 & ;si\ x >= Xs \end{cases}$$

$$Z=\begin{cases} (1/Xs)x+1 & ;si\ -Xs <= x < 0 \\ (-1/Xs)x+1 & ;si\ 0 <= x <= Xs \\ 0 & ;si\ x < -Xs\ o\ x > Xs \end{cases}$$

El parámetro Xs proporciona una forma de variar el punto de intersección de las funciones de membresía, permitiendo flexibilidad para la sintonización.

Las reglas utilizadas y mostradas en la matriz de la Fig. 2.13 son las siguientes:

R1: Si e(k) es negativo y Δe(k) es negativo entonces Δu(k) es negativa.
R2: Si e(k) es zero y Δe(k) es negativo entonces Δu(k) es negativa.
R3: Si e(k) es positivo y Δe(k) es negativo entonces Δu(k) es zero.
R4: Si e(k) es negativo y Δe(k) es zero entonces Δu(k) es negativa.
R5: Si e(k) es zero y Δe(k) es zero entonces Δu(k) es zero.
R6: Si e(k) es positivo y Δe(k) es zero entonces Δu(k) es positivo.
R7: Si e(k) es negativo y Δe(k) es positivo entonces Δu(k) es zero.
R8: Si e(k) es zero y Δe(k) es positivo entonces Δu(k) es positivo.
R9: Si e(k) es positivo y Δe(k) es positivo entonces Δu(k) es positivo.

Las reglas anteriores aplican una corrección de acuerdo al signo de las dos variables de entrada; sin embargo, no es la única configuración posible, éstas pueden ser modificadas con el fin de mejorar el desempeño como se hará en la fase de sintonización.

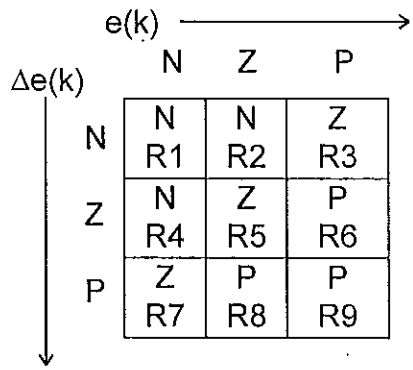


Fig. 2.13. Matriz de reglas.

e) Método de inferencia para el cálculo de la acción de control.

Las reglas enunciadas anteriormente y mostradas en la Fig. 2.13 tienen la forma general:

$$Si\ U_1\ es\ B_{i1}\ \ y\ U_2\ es\ B_{i2}\ \ entonces\ \ V\ es\ D_i$$

Donde U_1 y U_2 son variables de entrada y V es la variable de salida; B_{i1} y B_{i2} son funciones de membresía de entrada y D_i de salida. El algoritmo

utiliza un proceso de *fusificación*, *inferencia* y *defusificación*, como el descrito en el Capítulo 1.

Como el tiempo de procesamiento es crucial para nuestro sistema, se decidieron usar funciones de salida tipo *Singleton* (es decir se utilizó el método de inferencia TVFI descrito en el Capítulo 1) para facilitar el cálculo del valor defusificado mediante la evaluación del centroide, de forma tal que cada función de membresía utilizada para la salida es representada por un sólo valor; así, las funciones mostradas en la Fig. 2.12 para las variables de entrada y salida, son transformadas a las funciones *Singleton* de la Fig. 2.14 para ser usadas sólo con la variable Δu(k).

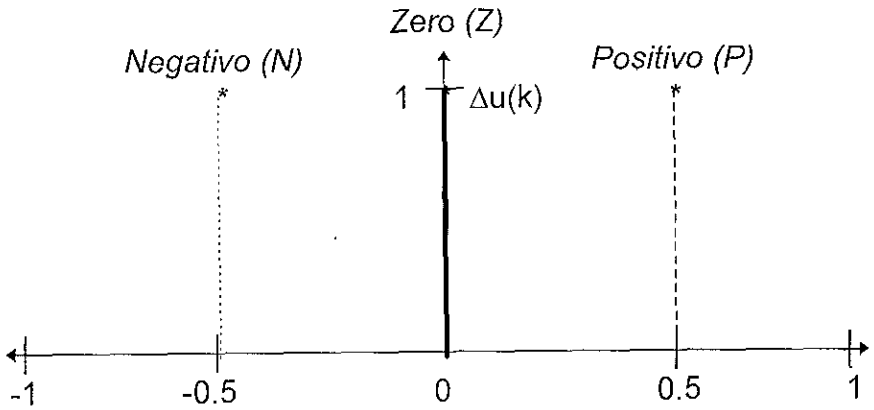


Fig. 2.14. Funciones de membresía Singleton para Δu(k), K_{du} = 128.

2.3.2.3 Sintonización del CLB-PI.

El CLB-PI se programó con las funciones de membresía descritas, así como para los valores de normalización indicados. A partir de este momento, el paso siguiente es la sintonización, que no resulta ser un problema trivial. Esta sintonización es posible efectuarla con base en los parámetros de normalización y de intersección de las funciones de membresía. Originalmente estos valores son establecidos a: K_e=1/64, K_d=1/128 , K_{du}=128 y X_s=0.5.

De acuerdo con la referencia [12] valores pequeños para K_e y K_d ocasionan un incremento en el error y en su variación respectivamente, mientras valores elevados disminuyen el error y su variación pero aumenta la inestabilidad. K_{du} afecta directamente la ganancia del CLB-PI; no obstante, valores muy elevados producen inestabilidad y valores pequeños producen respuestas lentas. El parámetro X_s afecta directamente la rapidez de respuesta y la estabilidad, a valores pequeños tiene una respuesta rápida pero inestable, mientras que a valores grandes disminuye la velocidad de respuesta mejorando la estabilidad, el valor recomendado para este parámetro es de 0.5.

a) *Variación individual de los parámetros de normalización.*

En esta sección, las variaciones de los parámetros se efectuaron para movimientos del objeto sobre una trayectoria constante de un pixel y período (*T_objeto*) de 15 centésimas.

La Fig. 2.15 presenta la variación del parámetro de normalización K_e para los valores 1/64, 1/84 y 1/100, se puede observar que el decremento de este parámetro mejora la estabilidad.

La Fig. 2.16 presenta la variación del parámetro de normalización K_d para los valores 1/90, 1/160, 1/200, se percibe que el error en la respuesta para valores pequeños de éste parámetro es menor, permitiendo que se ajuste mejor a la referencia.

La Fig. 2.17 presenta la variación del parámetro de normalización K_{du} para los valores 80, 95 y 150, es claro que valores pequeños mejoran directamente la estabilidad aunque existe error. Esta mejora es evidente dado que el parámetro está afectando directamente la ganancia del CLB-PI.

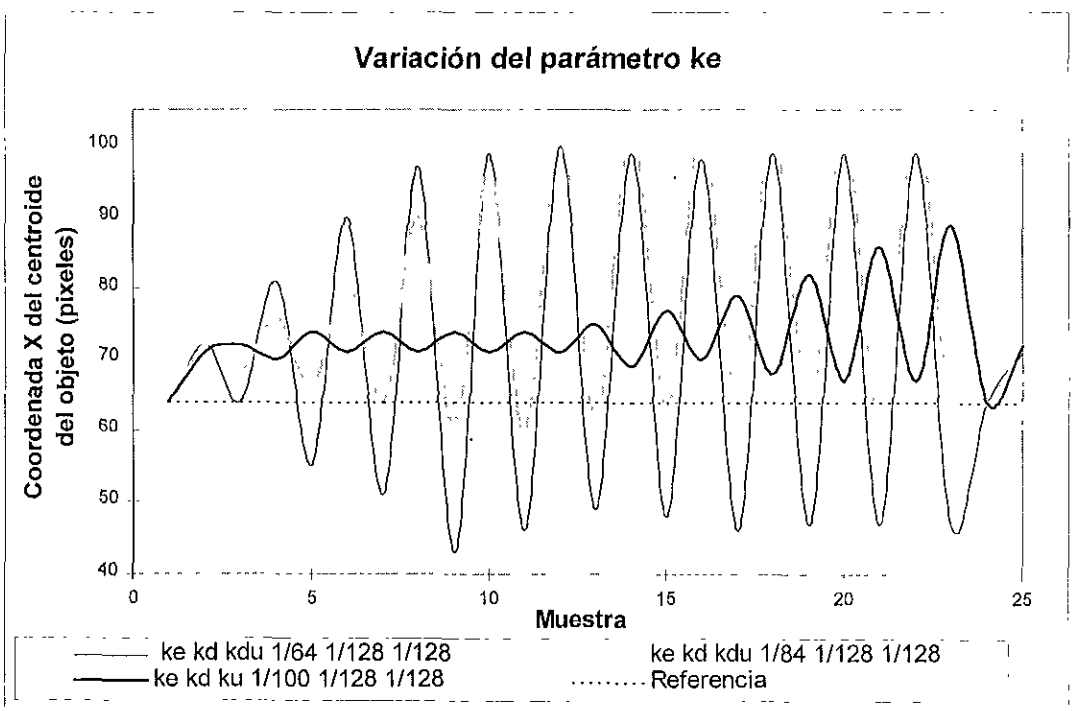


Fig. 2.15. Variación del parámetro de normalización K_e

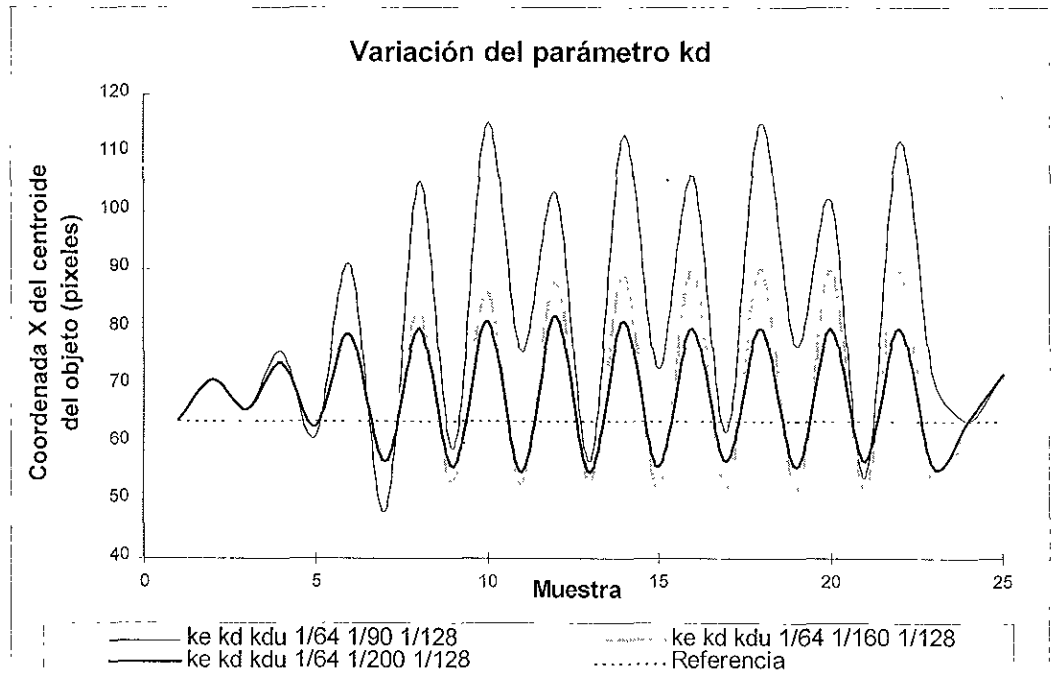


Fig. 2.16. Variación del parámetro de normalización K_d

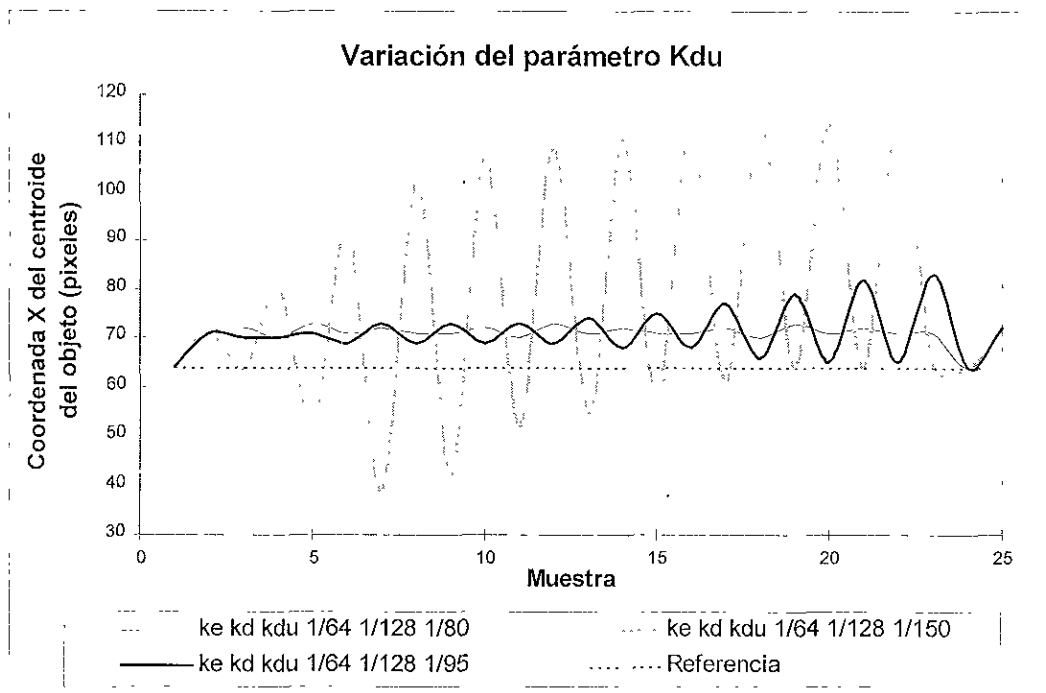


Fig. 2.17. Variación del parámetro de normalización K_{du}

La Fig. 2.18 presenta la variación del parámetro X_s para los valores 0.3, 0.5 y 0.8. Se observa como el efecto es muy parecido al producido por el parámetro K_{du}, de forma tal que valores grandes mejoran la estabilidad, mientras valores pequeños la empeoran. Esto es consecuencia de que un valor grande para X_s produce una mayor cobertura de la función de membresía *Z*, ocasionando respuestas más suaves; sin embargo, valores muy cercanos a 1 no son recomendables porque es equivalente a utilizar sólo una función de membresía.

b) Variación conjunta de los parámetros de normalización.

Se variaron en forma conjunta los parámetros de normalización tomando como referencia las variaciones individuales de la sección anterior. Estas sintonizaciones se efectuaron sobre la misma trayectoria recta horizontal con distribución unitaria y *T_objeto* = 15 centésimas.

La Fig. 2.19 muestra cuatro diferentes respuestas para diversos valores de los parámetros de sintonización. Como se observa la combinación K_e, K_d , K_{du} y X_s con valores 1/40, 1/180, 70 y 0.5 proporciona una respuesta con poco error pero con oscilaciones. Por otro lado, la combinación K_e, K_d , K_{du} y X_s con valores 1/80, 1/140, 100 y 0.5 proporciona una respuesta con mayor error pero con oscilaciones más pequeñas. Estas dos respuestas son las más aceptables.

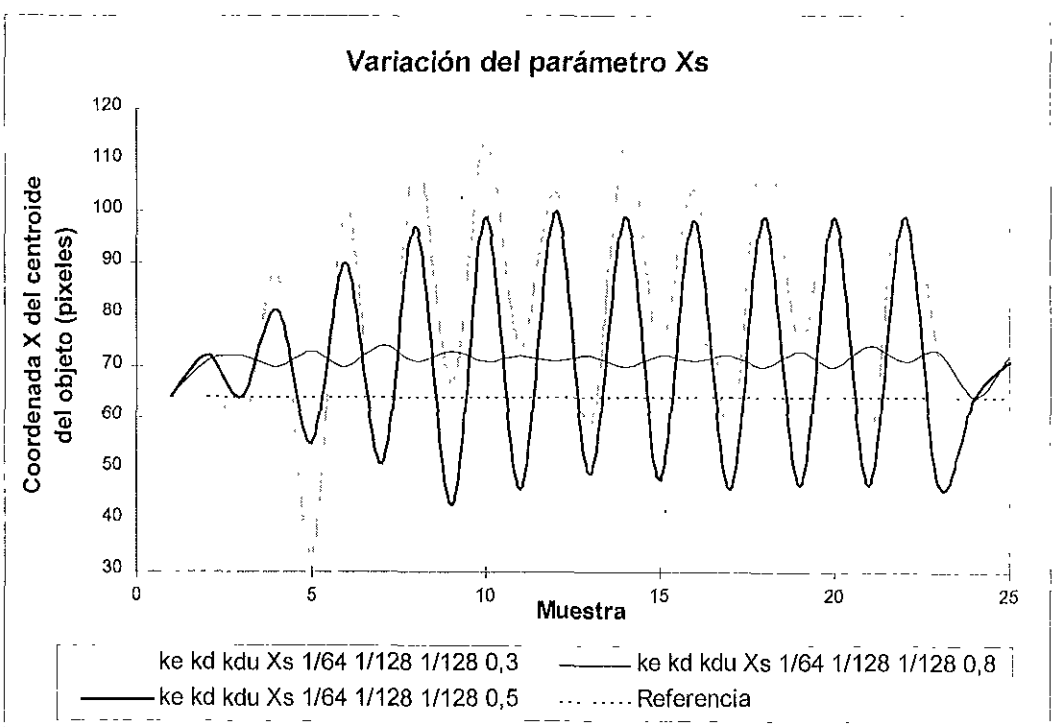


Fig. 2.18. Variación del parámetro de normalización X_s

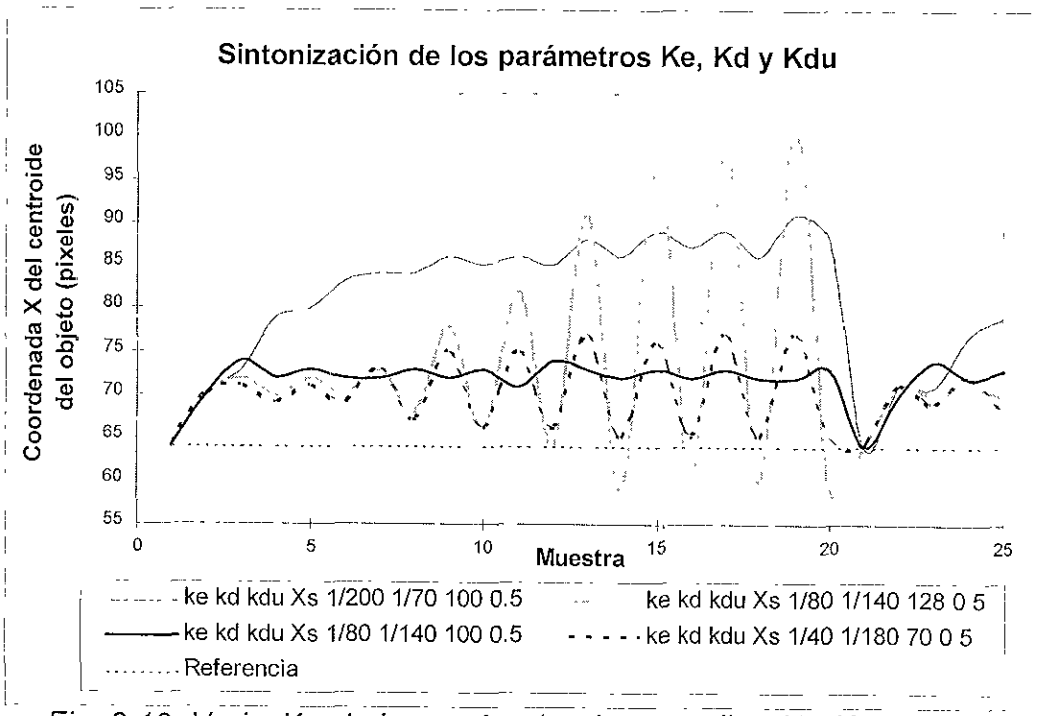


Fig. 2.19. Variación de los parámetro de normalización K_e , K_d y K_{du}

La Fig. 2.20 muestra otro conjunto de respuestas, una de ellas varia el valor del parámetro Xs. La mejor respuesta está constituida por la combinación K_e, K_d, K_{du} y X_s con valores : 1/40, 1/260, 70 y 0.5, tal conjunto se asemeja a las mejores respuestas de la Fig. 2.19.

c) Obtención de la mejor sintonización.

Las respuestas ilustradas desde la Fig. 2.15 a la Fig. 2.20 permitieron obtener el conjunto de parámetros K_e, K_d, K_{du}, X_s con valores 1/40, 1/260, 70 y 0.5 como la mejor combinación. No obstante, estos parámetros fueron solamente ajustados para una trayectoria horizontal con distribución unitaria, por lo tanto fue necesario evaluar los mismos parámetros con el conjunto de trayectorias mostradas en la Fig. 2.1.

La Fig. 2.21 muestra los resultados para las cuatro diferentes trayectorias; se observa que la respuesta para la trayectoria constante de un pixel es estable y tiende al valor de referencia. Esto es de esperarse dado que fue la trayectoria utilizada para establecer los valores de los parámetros de sintonización; sin embargo, la respuesta para las demás trayectorias es inestable y el objeto es perdido. Esto es consecuencia de valores pequeños para los parámetros K_{du} (ganancia) y K_d, que producen un incremento considerable en la variación del error.

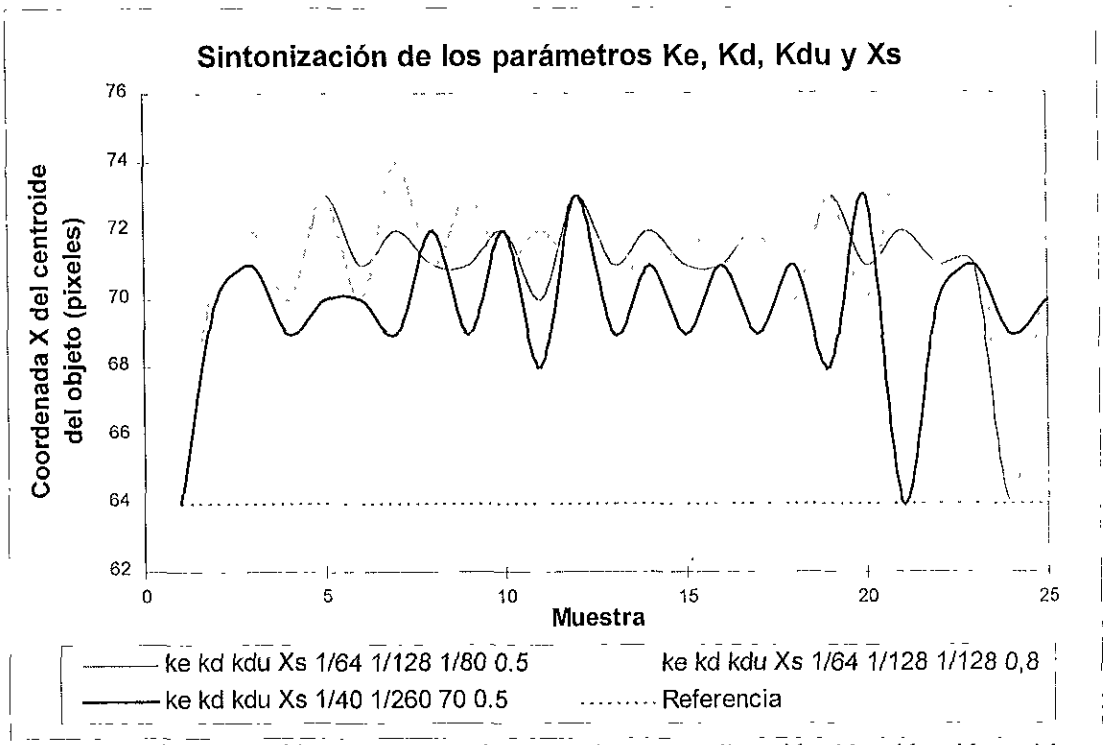


Fig. 2.20. Variación de los parámetro de normalización K_e , K_d , K_{du} y Xs

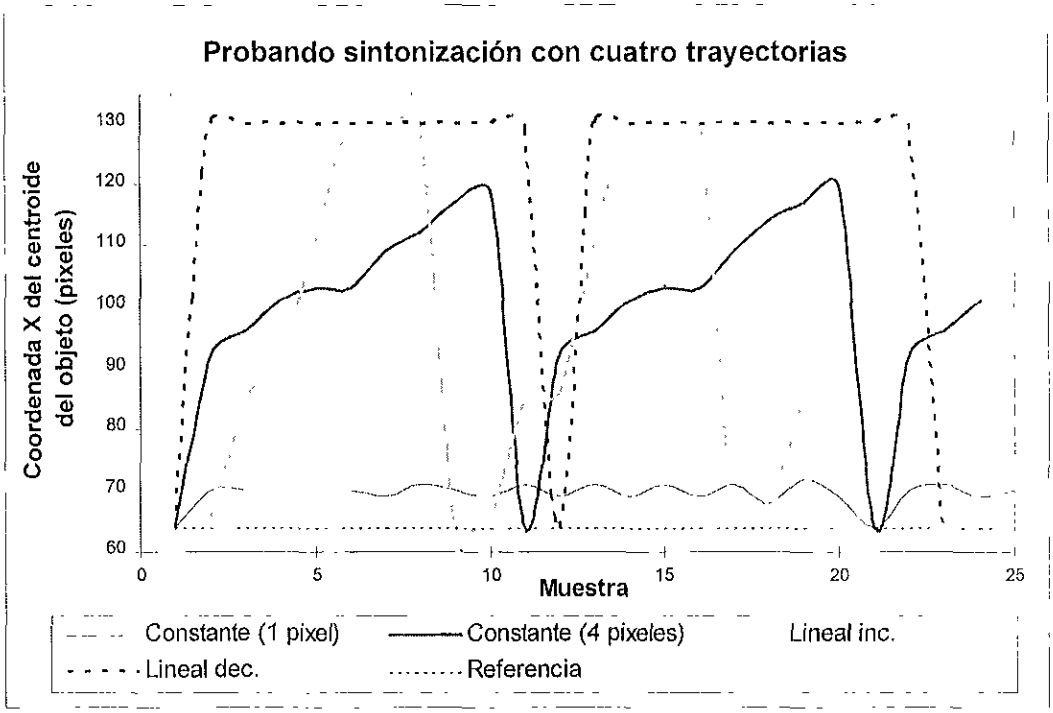


Fig. 2.21. Probando los parámetros de normalización para cuatro trayectorias diferentes.

d) Re-sintonización de los parámetros y de las reglas.

Las respuestas obtenidas en la Fig. 2.21 para las diversas trayectorias son no satisfactorias, a excepción de la trayectoria constante con densidad de un pixel, esto significa que los parámetros no son los óptimos.

Se incrementó la ganancia del CLB-PI a 145 para obtener respuestas más drásticas, el valor de K_d a 1/135 para contrarrestar la variación del error, K_e se disminuyó a 1/74 además de sintonizar dos reglas, para mejorar la estabilidad. La REGLA 2 cambió su consecuente de *N* a *Z* y la REGLA 8 de *P* a *Z*, en otras palabras cuando el error es próximo a cero no es conveniente aplicar cambios drásticos, evitando así oscilaciones.

La Fig. 2.22 ilustra los resultados obtenidos al efectuar los cambios comentados. Se observa claramente una mejora sustancial en la trayectoria constante con incrementos de 4 pixeles, sin perjudicar la respuesta con incrementos de 1 pixel. En las dos trayectorias restantes el objeto aún es perdido, pero esto es consecuencia de un valor pequeño *T_objeto=15* centésimas, que produce un colapso en la rutina de procesamiento de la imagen. Por tanto, se selecciona un valor *T_objeto = 25* centésimas para las gráficas desde la Fig. 2.23 en adelante. Este valor va a permitir apreciar las mejoras obtenidas por las modificaciones que se irán incorporando al CLB-PI. También va a permitir una comparación directa con la técnica de Corrección Simple del Error que utiliza *T_objeto = 25* centésimas.

La Fig. 2.23 ilustra los resultados para los mismos parámetros de la gráfica de la Fig. 2.22 pero con *T_objeto=25 centésimas*. Se observa como todas las trayectorias son seguidas y el objeto no es perdido, incluso para las trayectorias lineales crecientes y decrecientes.

2.3.2.4 Predictor de velocidad.

Las respuestas de la Fig. 2.23 oscilan considerablemente para los casos de las trayectorias lineales, tanto crecientes como decrecientes, además de que las respuestas están fuera de la referencia.

Esto último sucede porque la velocidad del objeto no sigue un patrón constante. Este problema se resuelve monitoreando y sumando algebraicamente la velocidad anterior del objeto a la acción de control proporcionada por el CLB-PI. Esto equivale a sumar un offset dependiente de la velocidad anterior, que se comporta como un predictor de velocidad, y se calcula como:

$$offset=Posición\_Actual - Posición\_Anterior \quad (2.3)$$

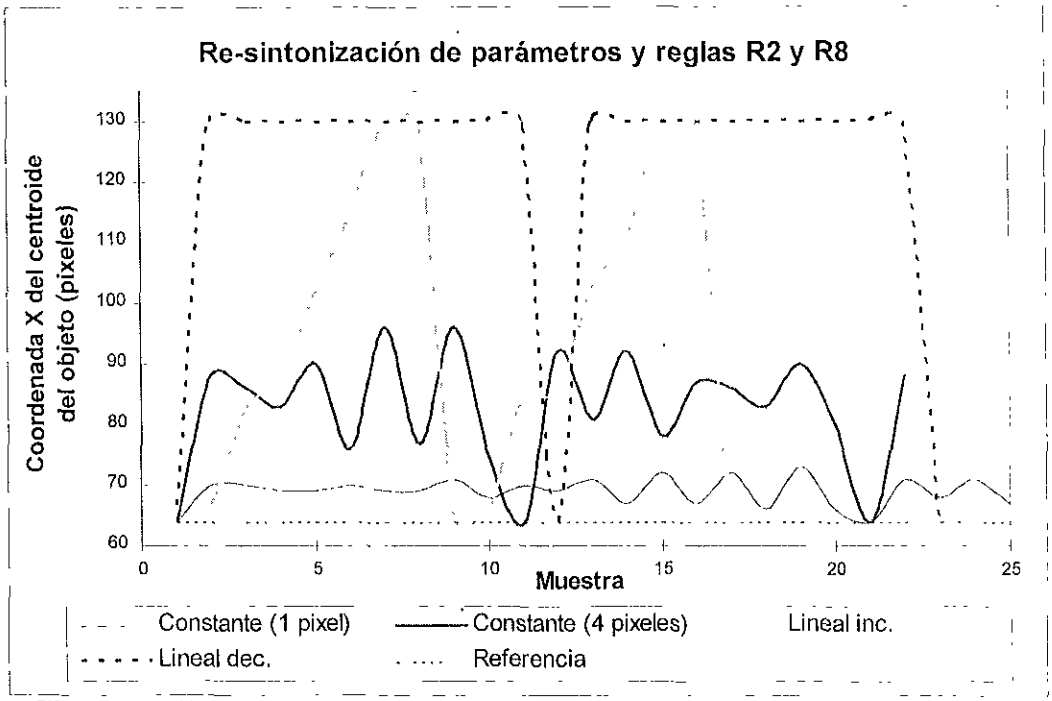


Fig. 2.22. Re-sintonización de los parámetros y de las reglas R2 y R8.

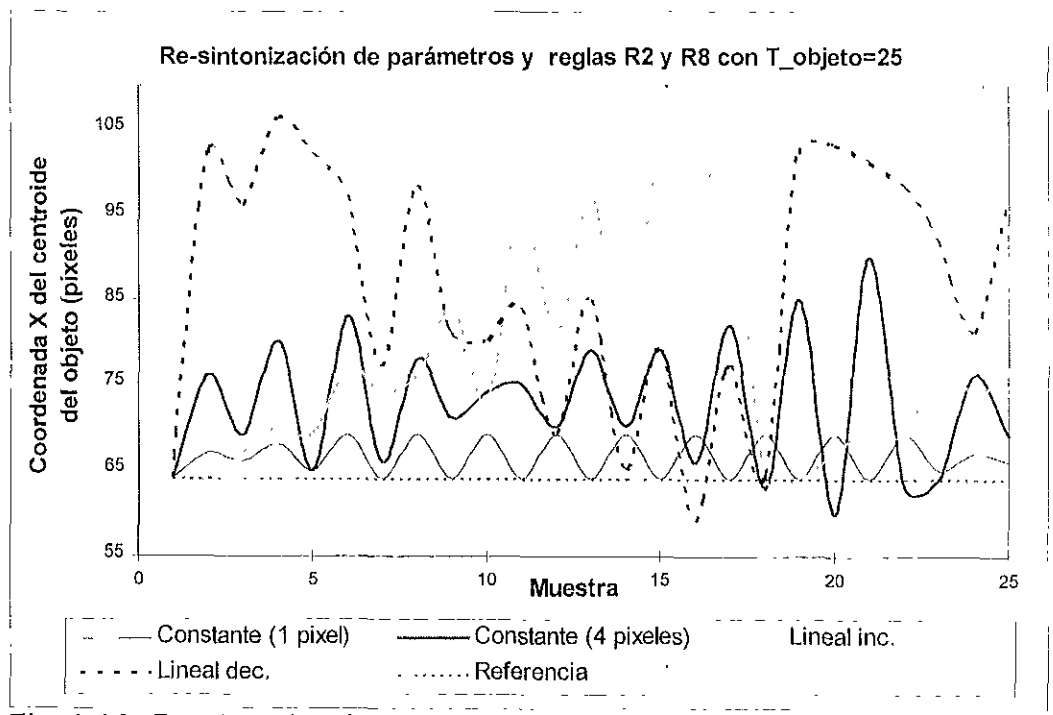


Fig. 2.23. Re-sintonización de los parámetros y de las reglas R2 y R8 con T_objeto=25 centésimas.

La expresión (2.3) es en realidad la velocidad actual del objeto en pixeles por tiempo de procesamiento, donde tiempo de procesamiento se refiere al tiempo requerido para la obtención de la posición del objeto. Por sus características de predictor de velocidad es llamado *offset de predicción* u *offset predecido*.

La respuesta debida a la incorporación de este offset se ilustra en la Fig. 2.24, para los mismos parámetros de sintonización y periodo de movimiento usados en la gráfica de la Fig. 2.23. Se observa que ahora las oscilaciones son alrededor de la referencia.

A partir de este momento, todas las gráficas incorporan a la acción de control dada por el CLB-PI la suma algebraica de éste offset.

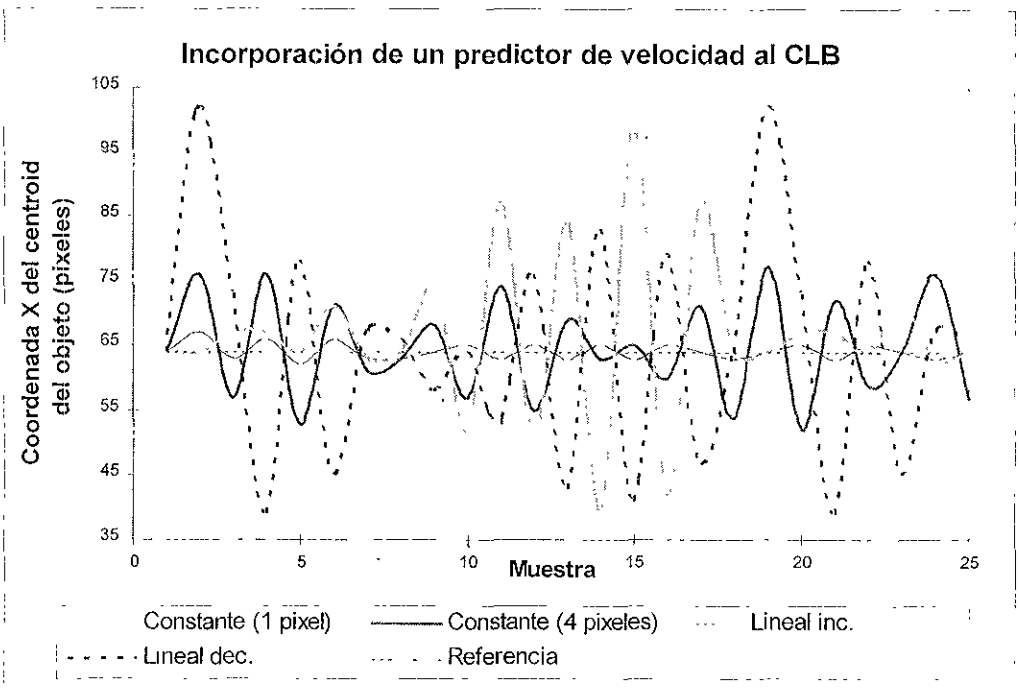


Fig. 2.24. Incorporación de un predictor de velocidad al CLB.

2.3.2.5 Incorporación de una variable de velocidad al CLB-PI.

La gráfica de la Fig. 2.24 muestra que el objeto no es perdido para ninguna de las trayectorias; sin embargo, existen oscilaciones bruscas en la mayoría de las respuestas a pesar de la sintonización de los parámetros y las reglas R2 y R8. Estas oscilaciones son producto de la rápida respuesta del sistema. Si la corrección efectuada en el estado actual sobrepasa la referencia, ocasiona que la siguiente corrección sea efectuada en sentido contrario, aún cuando el propio desplazamiento del objeto tienda a efectuar la corrección.

El problema se resuelve incorporando la velocidad del objeto al conjunto de variables, cuyo objetivo será restringir la corrección en sentido opuesto al desplazamiento del objeto en función de su velocidad. Esto evitará que un sobrepaso en la corrección ocasione un cambio drástico en el siguiente estado. La variable de velocidad se calcula exactamente igual al *offset predecido* de la ecuación (2.3). A partir de este momento se hará referencia al Controlador Lógico Borroso únicamente como CLB, dado que ahora tenemos una nueva variable de entrada que es la velocidad del objeto.

La matriz de reglas ahora puede generar un espacio tridimensional; sin embargo, siguen siendo nueve reglas utilizadas, donde cuatro de ellas dan importancia a la nueva variable, mientras las cinco restantes permanecen sin cambio. Se incorporan dos funciones de membresía llamadas *Negativo_L y Positivo_L*, cuyo objetivo es minimizar o maximizar la acción de control en función de la velocidad del objeto. Las reglas, obtenidas con base en las proposiciones de la Tabla 4, son:

R1: Si e(k) es negativo y Δe(k) es negativo y v(k) es Positivo_L entonces Δu(k) es negativa.
R2: Si e(k) es zero y Δe(k) es negativo entonces Δu(k) es zero
R3: Si e(k) es positivo y Δe(k) es negativo entonces Δu(k) es zero.
R4: Si e(k) es negativo y Δe(k) es zero y v(k) es Positivo_L entonces Δu(k) es negativa.
R5: Si e(k) es zero y Δe(k) es zero entonces Δu(k) es zero.
R6: Si e(k) es positivo y Δe(k) es zero y v(k) es Negativo_L entonces Δu(k) es positivo.
R7: Si e(k) es negativo y Δe(k) es positivo entonces Δu(k) es zero.
R8: Si e(k) es zero y Δe(k) es positivo entonces Δu(k) es zero.
R9: Si e(k) es positivo y Δe(k) es positivo y v(k) es Negativo_L entonces Δu(k) es positivo.

Tabla 4

Condición Centroide	e(k)	Δe(k)	Condición velocidad	Acción de Control	Reglas afectadas
Xo<Xw	(+)	cero	V(k)<0	Maximizar	R6
Xo<Xw	(+)	(+)	V(k)<0	Maximizar	R9
Xo<Xw	(+)	cero	V(k)>0	Minimizar	R6
Xo<Xw	(+)	(+)	V(k)>0	Minimizar	R9
Xo<Xw	(+)	(-)	no importa	Cero	R3
Xo>Xw	(-)	cero	V(k)>0	Maximizar	R4
Xo>Xw	(-)	(-)	V(k)>0	Maximizar	R1
Xo>Xw	(-)	cero	V(k)<0	Minimizar	R4

Xo>Xw	(-)	(-)	V(k)<0	Minimizar	R1
Xo>Xw	(-)	(+)	no importa	Cero	R7

Tabla 4. Proposiciones para obtener las reglas.

En el conjunto de reglas se puede observar como R2, R5 y R8, no son modificadas porque no existe error, de manera similar R3 y R7 porque el error y su variación al ser de signos contrarios se compensan.

a) Rango de operación para la variable V(k).

Velocidades de 64 pixeles por tiempo de procesamiento significa que el objeto ha logrado salir del campo de visión, en consecuencia se puede definir el intervalo de operación de la variable V(k) en [-64, 64]. Normalizando al intervalo [-1, 1], se obtiene un parámetro de normalización K_v=1/64; en consecuencia la velocidad normalizada se obtiene como :

$$V^*(k) = K_v V(k)$$

b) Utilización de funciones lineales para las funciones de membresía Negativo_L y Positivo_L.

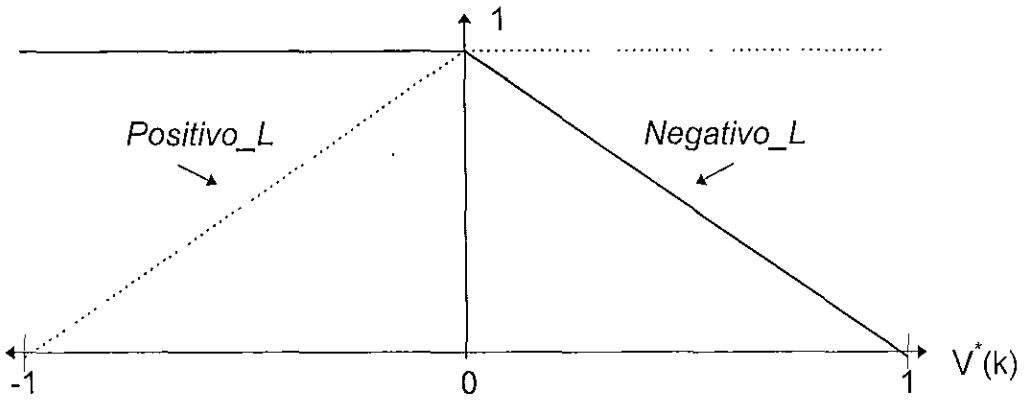
La Fig. 2.25 (a) muestra las funciones de membresía utilizadas para la variable V(k). La función *Negativo_L* inhibe el control entre más positiva sea la velocidad (movimiento del objeto a la derecha), la función *Positivo_L* inhibe el control entre más negativa sea la velocidad (movimiento del objeto a la izquierda).

El parámetro X_v permite modificar la forma de las funciones de la Fig. 2.25 (a) a la forma de la Fig. 2.25 (b), desplazando el punto en el cual la función comienza a tener valor de 1.

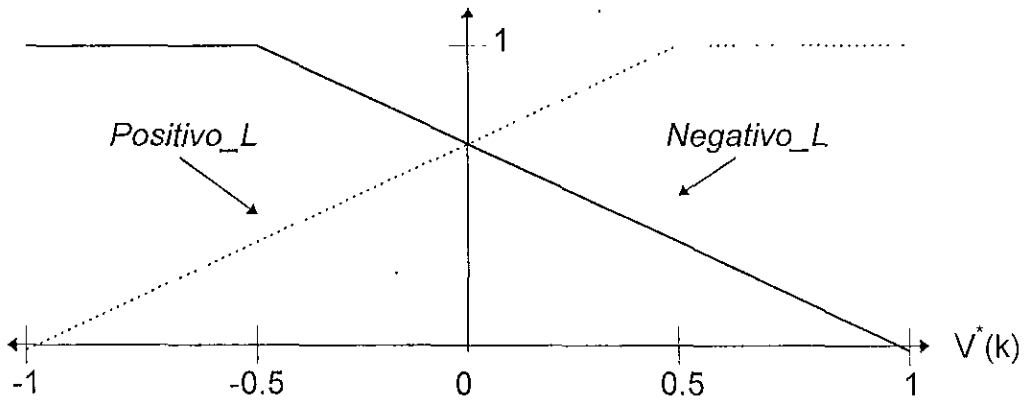
Las definiciones de las funciones de membresía *Positivo_L* y *Negativo_L* son las siguientes :

$$Negativo\_L = \begin{cases} (-x+1)/(1-X_V) & ;si\ X_V <= V^*(k) <= 1 \\ 0 & ;si\ V^*(k) > 1 \\ 1 & ;si\ V^*(k) < X_V \end{cases}$$

$$Positivo\_L = \begin{cases} (x+1)/(1-X_V) & ;si\ -1 <= V^*(k) <= -Xv \\ 0 & ;si\ V^*(k) < -1 \\ 1 & ;si\ V^*(k) > -X_V \end{cases}$$



(a) X_V=0.0



(a) X_V= -0.5

Fig. 2.25. Funciones de membresía para V(k).

La Fig. 2.26 ilustra las respuestas para valores de los parámetros K_v=1/64 y X_v=0.0, mientras la Fig. 2.27 ilustra las respuestas para valores K_v=1/64 y X_V= -1.0. Los restantes parámetros toman los valores que ya se sintonizaron.

Podemos notar las siguientes mejoras:

- La gráfica de la Fig. 2.27 muestra mejores resultados que la respuesta de la Fig. 2.26, principalmente en las trayectorias constante de 4 pixeles y lineal decreciente.

- Existe una mejora en la trayectoria lineal decreciente de la Fig. 2.27 en comparación con la Fig. 2.24.

De esta forma la respuesta para el caso de la Fig. 2.27 posee los parámetros: K_e=*1/74;* K_d= *1/135;* K_{du}= *145;* X_s = *0.5;* K_v= *1/64;* X_v = *-1.0*

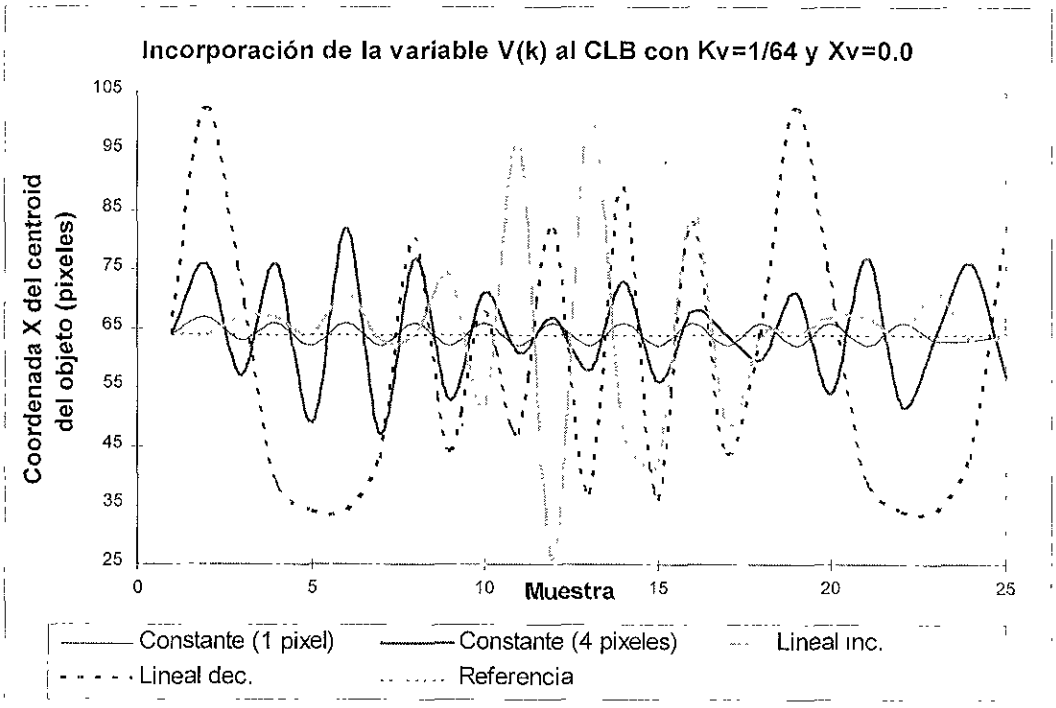


Fig. 2.26. Incorporación de la variable V(k) al CLB, con parámetros K_v=1/64 y X_v=0.0.

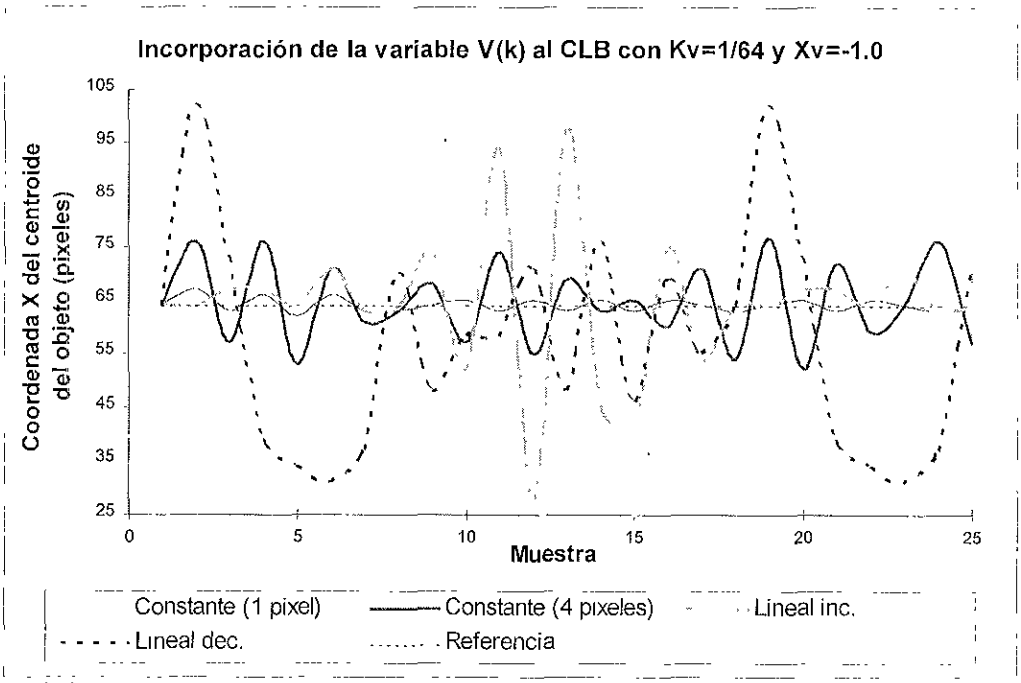


Fig. 2.27. Incorporación de la variable V(k) al CLB, con parámetros K_v=1/64 y X_v=-1.0.

c) Utilización de funciones exponenciales para las funciones de membresía Negativo_L y Positivo_L.

Se utilizaron las funciones de membresía exponenciales mostradas en la Fig. 2.28 para la variable de entrada velocidad $V^*(k)$. Se pretende que a mayores velocidades la caída de la acción de control sea drástica. Las funciones a utilizar están constituidas por una parte constante y otra exponencial. La parte exponencial es del tipo $y=\alpha\, e^{\pm\beta x}$; el parámetro α=1 es fijo debido a que el valor máximo de pertenencia es 1, y conforme el parámetro β se incrementa la caída es más drástica. A continuación se muestran las definiciones de las funciones *Negativo_L* y *Positivo_L*.

$$Negativo\_L=\begin{cases} e^{-\beta x} & ;\ \text{si } 0 <= V^*(k) <= 1 \\ 0 & ;\ \text{si } V^*(k) > 1 \\ 1 & ;\ \text{si } V^*(k) < 0 \end{cases} \qquad Positivo\_L=\begin{cases} e^{\beta x} & ;\ \text{si } -1 <= V^*(k) <= 0 \\ 0 & ;\ \text{si } V^*(k) < -1 \\ 1 & ;\ \text{si } V^*(k) > 0 \end{cases}$$

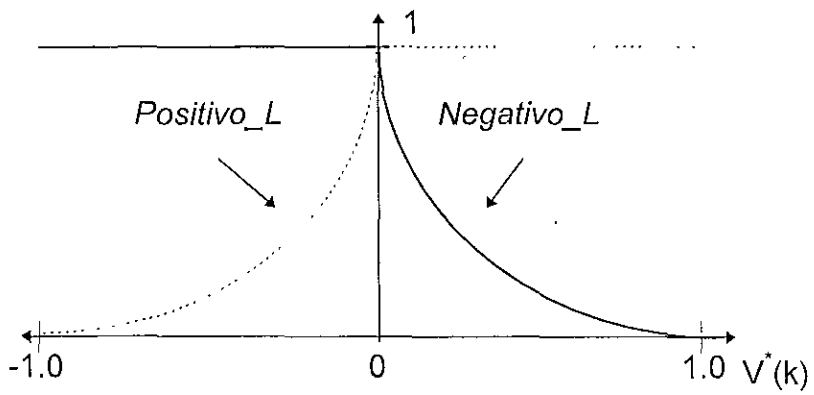


Fig. 2.28. Funciones de membresía para la variable de entrada $V^*(k)$

Las Figs. 2.29, 2.30, 2.31 y 2.32 muestran respuestas para diversos valores del parámetro β. Como se observa estas gráficas tienen mejor desempeño que la gráfica de la Fig. 2.27 para todas las trayectorias en dos aspectos:

- Disminuyen los picos superiores e inferiores para las trayectorias constantes y lineal creciente; en esta última, la mejora es muy significativa.

- La frecuencia de las oscilaciones disminuye para la trayectoria lineal decreciente y creciente.

La gráfica de la Fig. 2.31 para un valor de β = 8.0 posee el mejor desempeño, con el único inconveniente que a mayor valor de β la acción de control es más inhibida aún con velocidades pequeñas, de ahí que la respuesta para β = 10.0 no sea óptima.

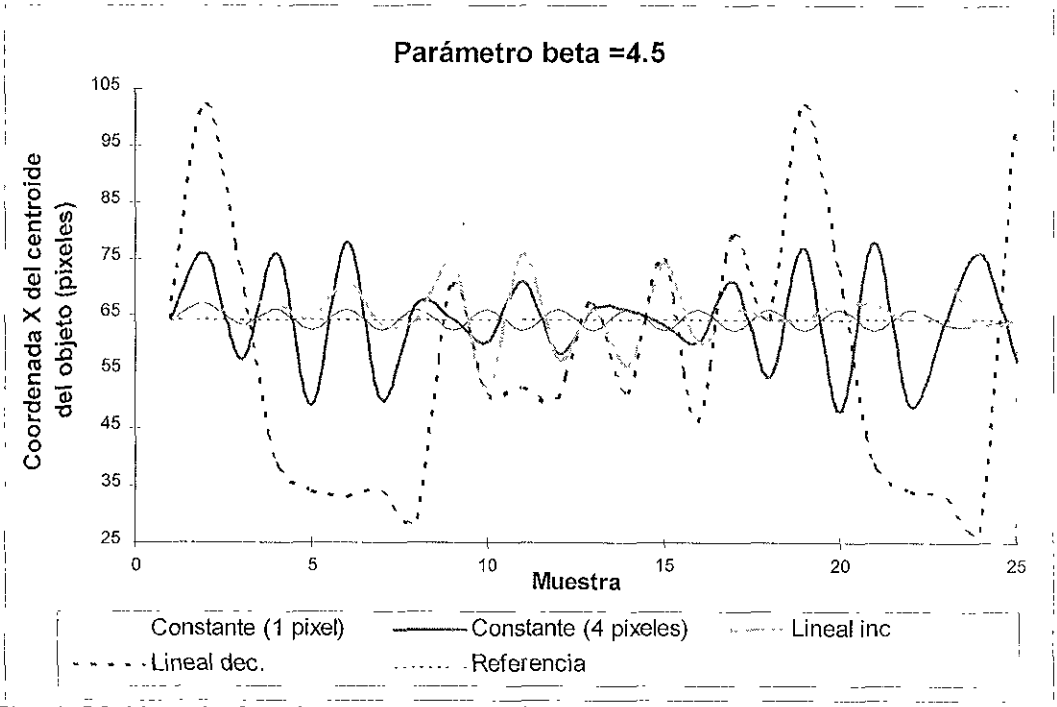


Fig. 2.29 Uso de funciones exponenciales para Positivo_L y Negativo_L con parámetro β = 4.5

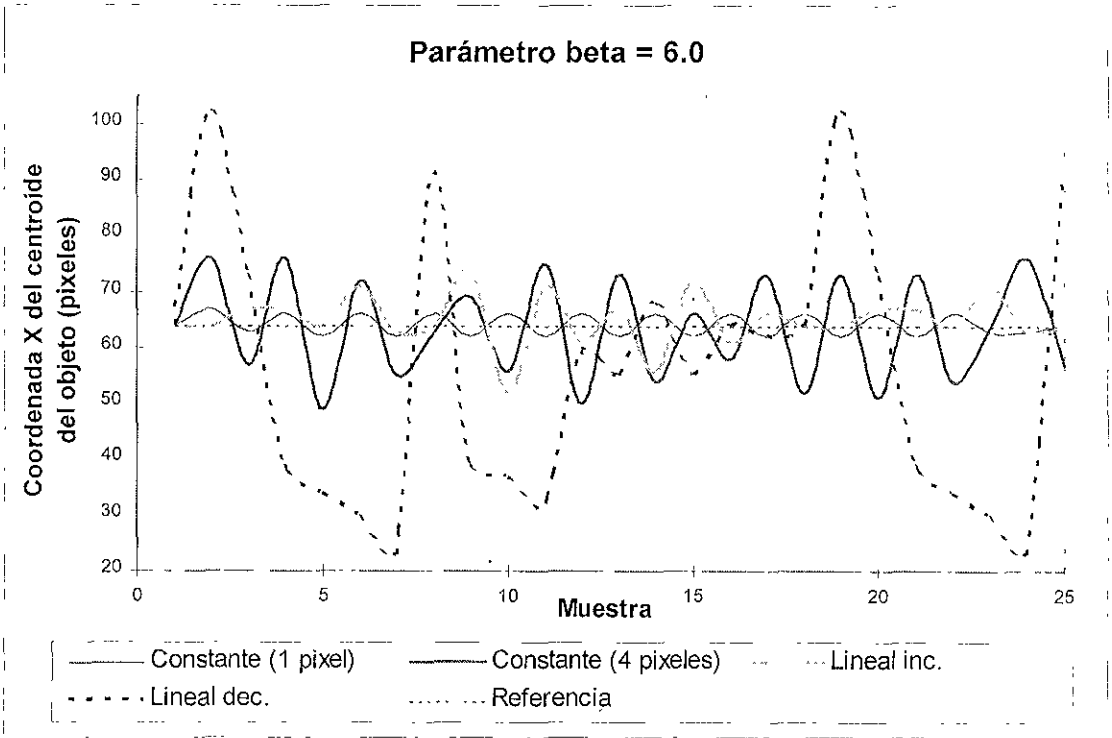


Fig. 2.30 Uso de funciones exponenciales para Positivo_L y Negativo_L con parámetro β = 6.0

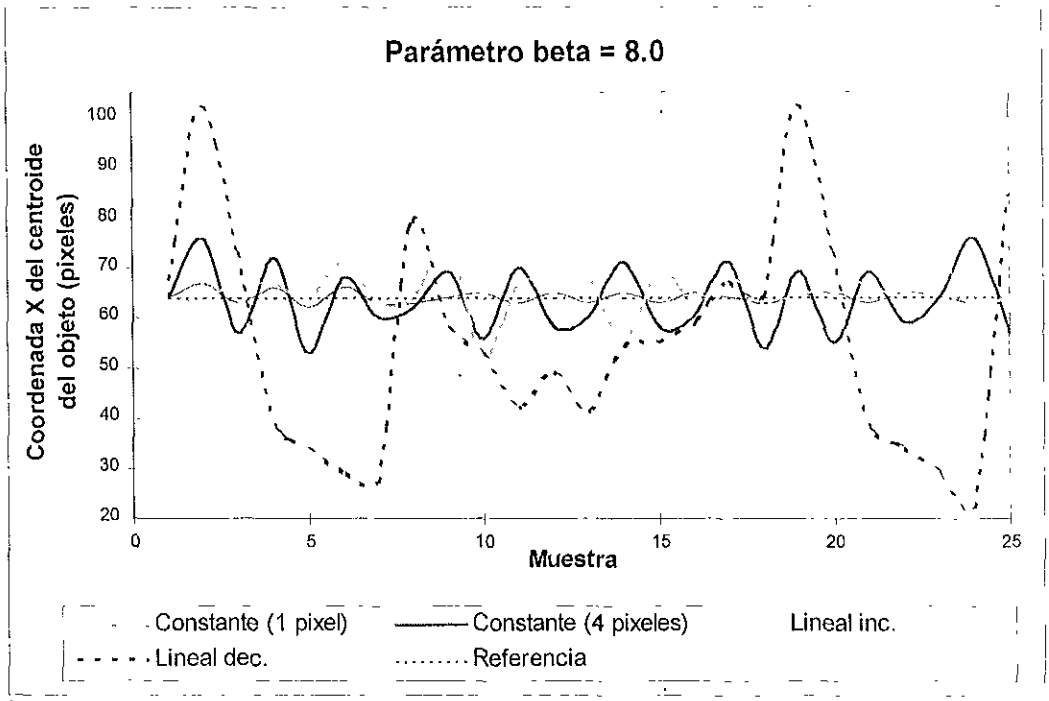


Fig. 2.31 Uso de funciones exponenciales para Positivo_L y Negativo_L con parámetro $\beta = 8.0$

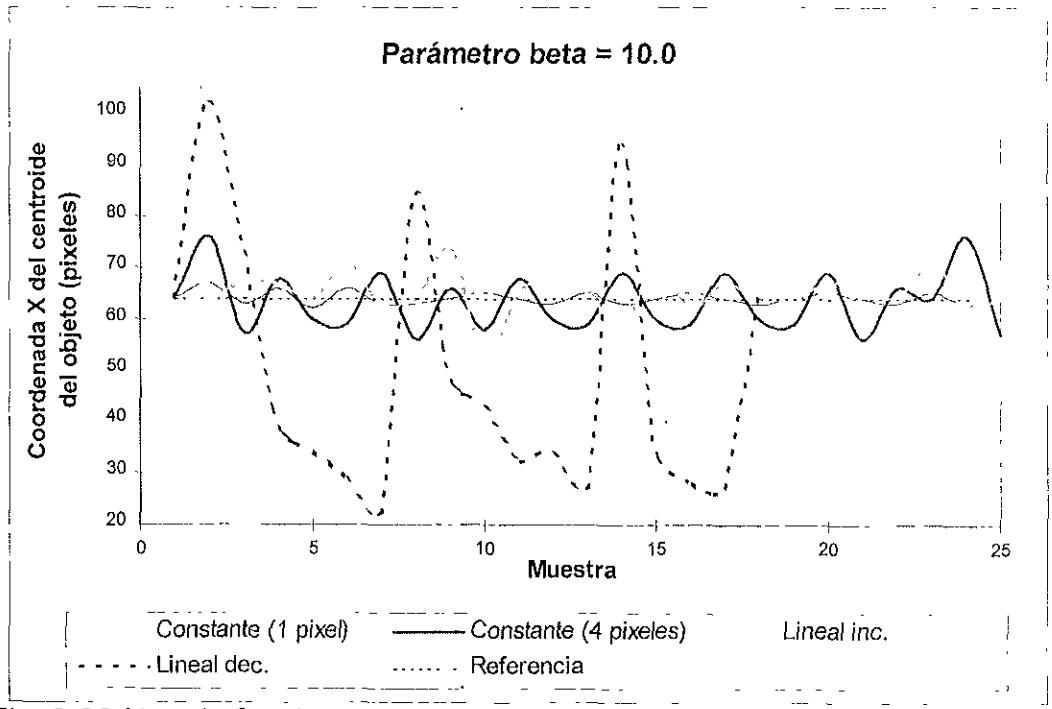


Fig. 2.32 Uso de funciones exponenciales para Positivo_L y Negativo_L con parámetro $\beta = 10.0$

2.3.2.6 Ajuste dinámico de la ganancia K_{du} para el CLB.

En la gráfica de la Fig. 2.31, para un valor de β = 8.0, se tiene el problema de picos para el caso lineal decreciente, esto es consecuencia del valor estático de nuestra ganancia K_{du}, ya que si la velocidad disminuye drásticamente aún se sigue aplicando la misma ganancia. Esto se corrige variando la ganancia dinámicamente en función de la velocidad V(k), de manera que a velocidades elevadas la ganancia sea mayor y, a velocidades menores disminuya.

2.3.2.6.1 Variación lineal de la ganancia K_{du} en función de la velocidad V(k).

La ganancia utilizada hasta este momento tiene un valor fijo de 145 que resulta muy elevado cuando la velocidad está disminuyendo; por consiguiente, es posible considerar un número cercano a 145 como límite superior; como límite inferior no es posible utilizar cero porque ganancias pequeñas producen respuestas lentas. El límite inferior es considerado aproximadamente igual a la mitad del límite superior. La Fig. 2.33 ilustra la función utilizada para hacer variar la ganancia K_{du}.

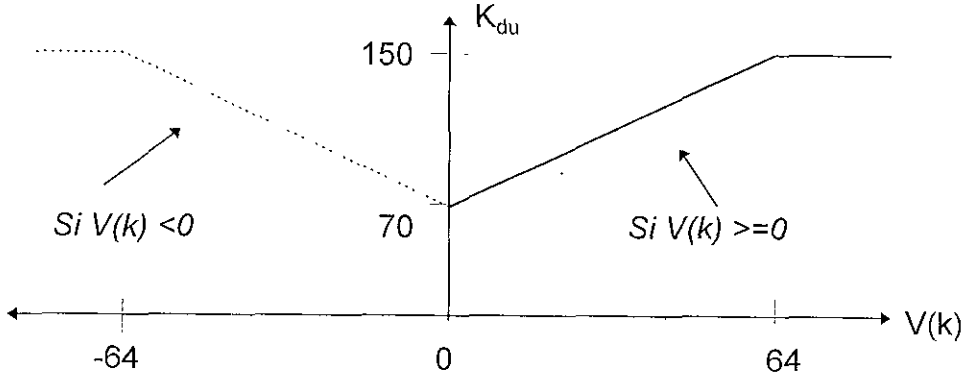


Fig. 2.33 Variación lineal de la ganancia K_{du}.

Las funciones de la Fig. 2.33 están definidas como:

$$K_{du} = \begin{cases} 150 & ; \text{si } V(k) > 64 \\ (80/64)x + 70 & ; \text{si } 0 <= V(k) <= 64 \\ 150 & ; \text{si } V(k) < -64 \\ (-80/64)x + 70 & ; \text{si } -64 <= V(k) < 0 \end{cases}$$

La Fig. 2.34 ilustra el desempeño del CLB. Se observa una mejora significativa en los picos producto del sobretiro en la acción de control, no sólo para la trayectoria lineal decreciente, sino para todas las trayectorias. Esto es consecuencia de adaptar la ganancia según las condiciones de velocidad.

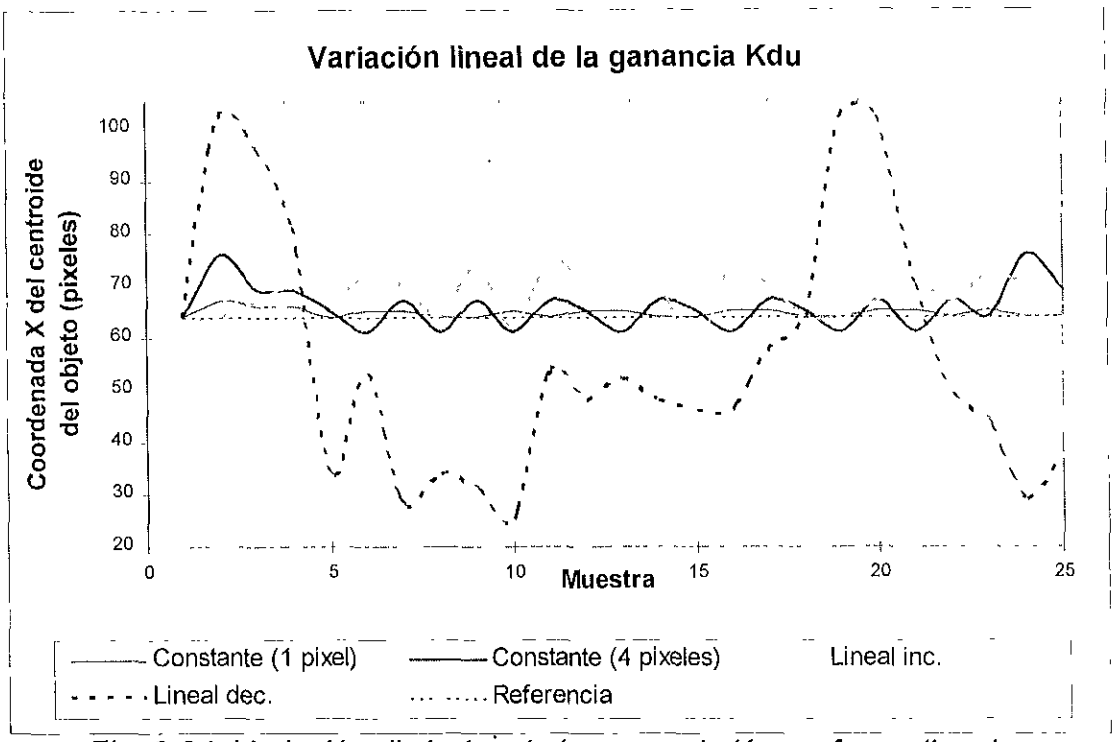


Fig. 2.34. Variación dinámica de la ganancia K_{du} en forma lineal.

2.3.2.6.2 Control borroso de la ganancia K_{du} en función de la velocidad V(k).

Se incorporó un subsistema borroso encargado de obtener la ganancia K_{du} en función de la velocidad V(k) y la aceleración A(k)= V(k) - V(k-1).

En la Fig. 2.35 se muestra el diagrama a bloques del sistema propuesto para la variación de la ganancia K_{du}.

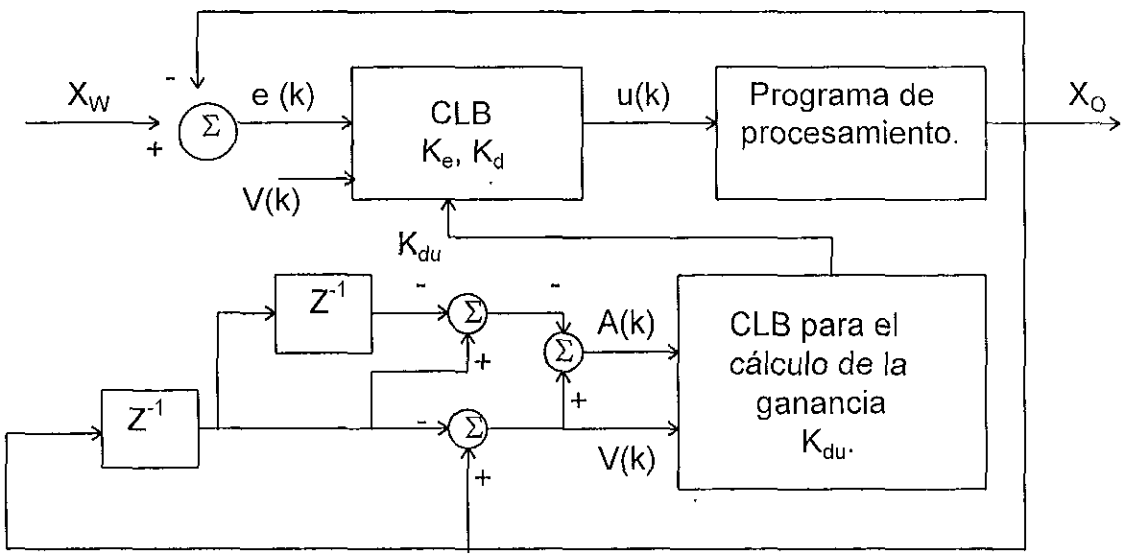


Fig. 2.35. Variación de la ganancia K_{du} usando otro CLB.

a)Variables de entrada.

Tenemos las variables de entrada velocidad V(k) y aceleración A(k), como se muestra en la Fig. 2.35. El intervalo de operación para la variable V(k) se definió anteriormente en [-64, 64]; por consiguiente A(k) queda definido (por los extremos máximos y mínimos de la velocidad) en el intervalo [-128, 128].

b)Variable de salida.

La salida está constituida por el parámetro de ganancia K_{du} requerido por el sistema de control de posición de la cámara. El intervalo de operación para este parámetro es [40, 200], semejante al utilizado en la variación lineal del parámetro K_{du}.

c) Normalización de las variables de entrada y salida al intervalo [-1, 1].

- V(k) $\in$ [-64, 64] entonces K_v=1/64 por lo tanto $V^*(k)=K_v\ V(k)$.
- A(k) $\in$ [-128, 128] entonces K_{dv}= 1/128 por lo tanto $A^*(k) = K_{dv}\ A(k)$.
- $K^*_{du} \in$ [-1, 1], se desea que $K_{du} \in$ [40, 200] por lo tanto $Kdu=80K^*_{du}+120$.

d) Funciones de membresía y reglas difusas.

Se utilizaron las funciones de membresía ilustradas en la Fig. 2.12 y la matriz de reglas de la Fig. 2.36; estas reglas pretenden incrementar la ganancia cuando la velocidad o aceleración es *N* o *P*.

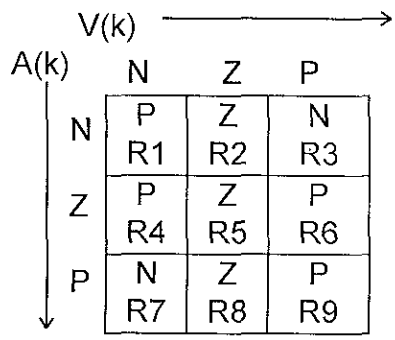


Fig. 2.36. Matriz de reglas.

e) Sintonización de los parámetros K_v, K_{dv}, X_s, y desnormalización para el cálculo de K_{du}.

La Fig. 2.37 muestra la mejor respuesta a una variación estratégica de los parámetros alrededor de sus valores por definición, de manera que la salida K_{du}

perteneciente al intervalo [40, 200] se calcula como K_{du}= 80K^*_{du}+120; los demás parámetros son establecidos como K_v=1/94, K_{dv} =1/60, X_s= 0.5.

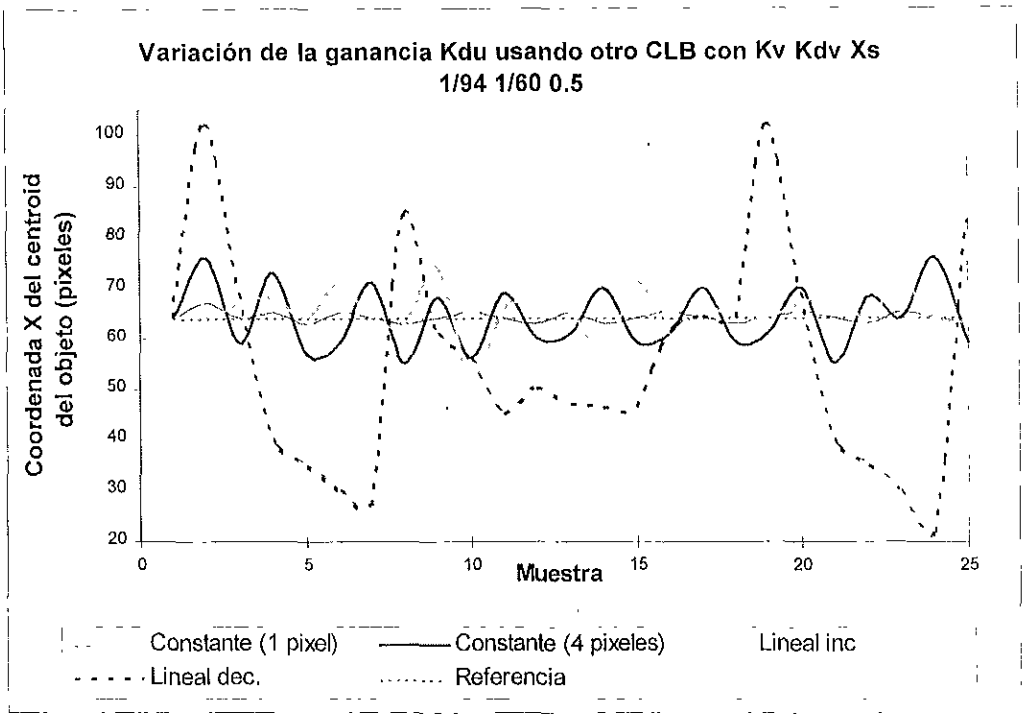


Fig. 2.37. Variación de la ganancia K_{du} usando otro CLB.

f) Sintonización de las reglas y de la expresión de desnormalización K_{du}=80K^*_{du}+120.

La gráfica de la Fig. 2.37 aún tienen los problemas de picos no deseados y oscilación; esto se resuelve sintonizando algunas de las reglas relativas a la estabilidad y, la ganancia de salida K_{du}=80K^*_{du}+120.

Las reglas R4 y R6 de la Fig. 2.36, para casos extremos de velocidad pero sin variación de la misma son sintonizadas de *P* a *Z* para evitar cambios bruscos. Las reglas R2 y R8 para velocidades relativamente bajas y aceleraciones drásticas pasan de *Z* a *N*, con el objeto de eliminar correcciones bruscas a velocidades bajas.

El nuevo intervalo de ganancia para calcular la salida K_{du} es [40, 150], sólo se sintoniza el límite máximo (200 anteriormente) a 150, de manera que la salida K_{du} se calcula como K_{du}=55K^*_{du}+95. La Fig. 2.38 ilustra la respuesta para este nuevo intervalo.

Las respuestas para el ajuste de la ganancia K_{du} usando otro CLB (ilustradas en la Fig. 2.38) en comparación, con las respuestas para el ajuste de la ganancia K_{du} en forma lineal (mostradas en la Fig. 2.34) permite observar:

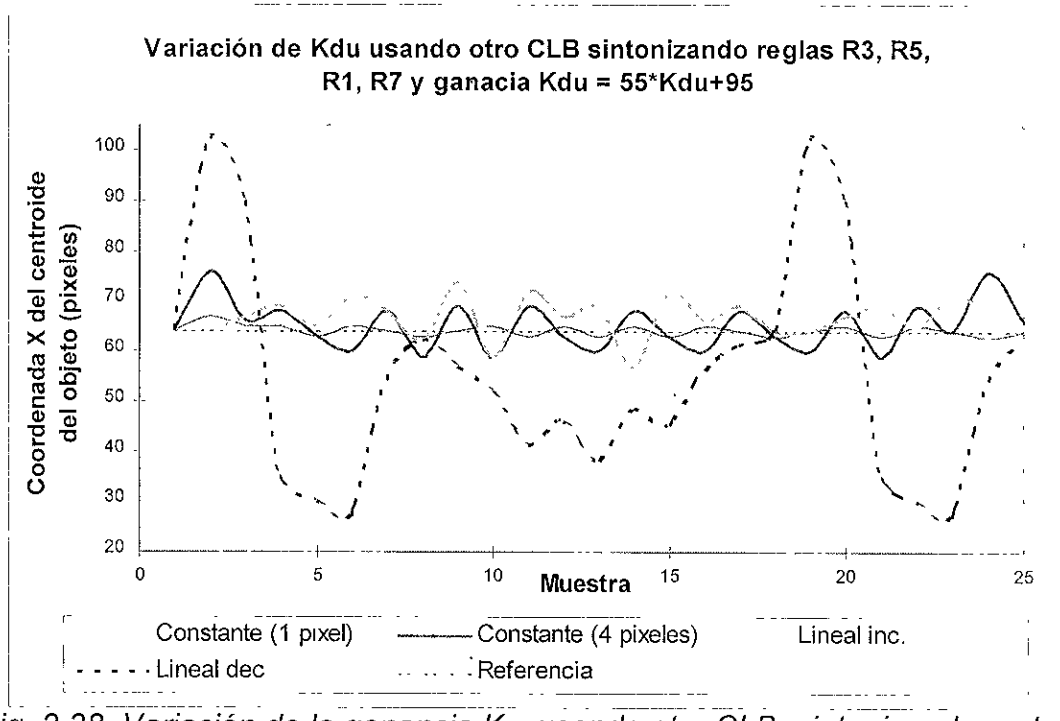


Fig. 2.38. Variación de la ganancia K_{du} usando otro CLB, sintonizando reglas y ganancia de salida.

- La respuesta para una trayectoria constante en la Fig. 2.34, presenta oscilaciones menos pronunciadas que la correspondiente en la Fig. 2.38; sin embargo, esta última si está ubicada en ambos lados de la referencia.

- En relación a la trayectoria lineal creciente, la Fig. 2.38 presenta mejor respuesta en contraste con la Fig. 2.34 dado que se ubica alrededor de la referencia. Sin embargo, para la trayectoria constante con incremento de 4 pixeles, la respuesta de la Fig. 2.38 presenta oscilaciones mayores que la correspondiente respuesta de la Fig. 2.34, pero poco significativas.

- La ventaja principal de la respuesta de la Fig. 2.38 sobre la Fig. 2.34, radica en que la trayectoria lineal decreciente es capaz de ajustarse más rápidamente a la referencia, sin presentar oscilaciones extremas.

Por las razones anteriores, el cálculo del parámetro K_{du} empleando otro CLB presenta las mejores respuestas para las trayectorias analizadas. Se utilizan los parámetros K_v = 1/94, K_{dv}= 1/60, X_s = 0.5 y, K_{du}=55K^*_{du}+95.

En síntesis, nuestro CLB proporciona una acción de control calculada con base en:

- El error y su variación.
- Predictor de velocidad (offset).

- Magnitud y sentido de la velocidad.
- Variación dinámica de la ganancia, controlada en función de la velocidad y aceleración del objeto, usando otro CLB.

Comparando los resultados de las técnicas de Corrección Simple del Error y Control Lógico Borroso mostradas en las Figs. 2.8 y 2.38 se observa, como la última técnica es más eficiente para todas las trayectorias analizadas, en los siguientes aspectos:

- Para ninguna de las trayectorias el objeto en movimiento es perdido.
- Todas las respuestas están ubicadas alrededor de la referencia.

Por tanto, el CLB presenta el mejor desempeño que supera notablemente la Corrección Simple del Error.

CAPÍTULO 3
IMPLANTACIÓN.

Este capítulo muestra el sistema de adquisición de imágenes utilizado, la estructura mecánica y diseño electrónico para el movimiento y control de la cámara, así como el software para la conjunción de los tres subsistemas que conforman el sistema físico completo para el seguimiento de un objeto en movimiento.

3.1 Sistema de Adquisición de Imágenes.

Este subsistema permite adquirir y digitalizar, mediante una computadora personal, una tarjeta digitalizadora y, con base en el estándar NTSC, las imágenes provenientes de una cámara de video CCD. También permite acceso completo a la información de los pixeles.

3.1.1 Cámara CCD RS-170.

La cámara utilizada es pequeña con el objetivo de que el tamaño y peso de la estructura mecánica y los motores utilizados para su movimiento sean mínimos y, en consecuencia abatir costos.

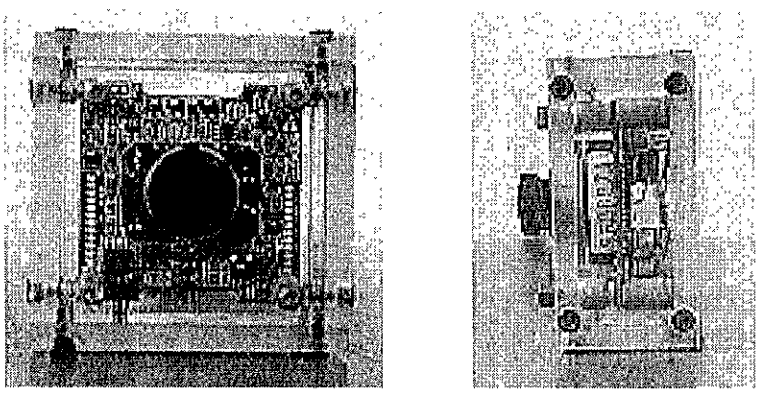
Dado que el color es irrelevante para el tipo de procesamiento efectuado, fue seleccionada una cámara de video analógica CCD blanco y negro RS-170. El modelo es MTV-361EM, sus características más importantes se muestran en la

Tabla 3.1. Se observa que es muy ligera y pequeña, con peso de 40 gramos y menos de 5 cm por lado. La Fig. 3.1 muestra la cámara utilizada.

Tabla 3.1

Característica.	Descripción.
Dispositivo sensor de imagen.	1/3 " CCD transferencia interlineado.
Área del sensor de imagen.	4.9 mm × 3.7 mm.
Elementos de estructura.	510(H) × 492(V).
Frecuencia horizontal.	15.750 KHz.
Frecuencia vertical.	60 Hz.
Sistema de barrido.	525 líneas, 60 campos/sec.
Salida de video.	1.0 Vpp, video compuesto, 75 ohm carga.
Ángulo de lente.	Diagonal 65 2' , Horizontal 52 1' , Vertical 39 5'.
Alimentación y consumo.	DC 12V, 100 mA, 1.2 W.
Dimensiones.	44mm × 44mm × 15mm.
Temperatura de operación.	-20 °C ~ +55 °C.
Peso.	40 g.

Tabla 3.1. Características de la Cámara CCD.



(a) (b)

Fig. 3.1. Cámara de video CCD utilizada. (a) Vista frontal y (b) Vista de perfil.

3.1.2 Frame Grabber Matrox Meteor.

El sistema requiere de una tarjeta digitalizadora capaz de transformar la señal analógica proveniente de la cámara en un formato digital sin compresión. El formato requerido es de tipo RAW, éste representa la tonalidad de cada pixel de la imagen como un valor entero dentro del intervalo [0, 255]. Esto permite que la imagen capturada sea utilizada directamente por el sistema de procesamiento. Además se requiere el resultado de la digitalización se traslade al sistema de

memoria principal de la PC. Por evaluación de costos y capacidades se seleccionó el Frame Grabber *Matrox Meteor*.

3.1.2.1 Adquisición

Matrox Meteor es un Frame Grabber para slot PCI. Puede hacer transferencia en tiempo real al sistema de memoria RAM de la PC. Posee capacidades para adquisición estándar en tonos de gris y color. Acepta los estándares RS-170/CCIR para señales en tonos de gris y los estándares NTSC/PAL para señales en color. La tarjeta dispone de cuatro entradas de video para interfaz con hasta cuatro cámaras. Las entradas son seleccionadas por software. El diagrama a bloques del Frame Grabber se ilustra en la Fig. 3.2.

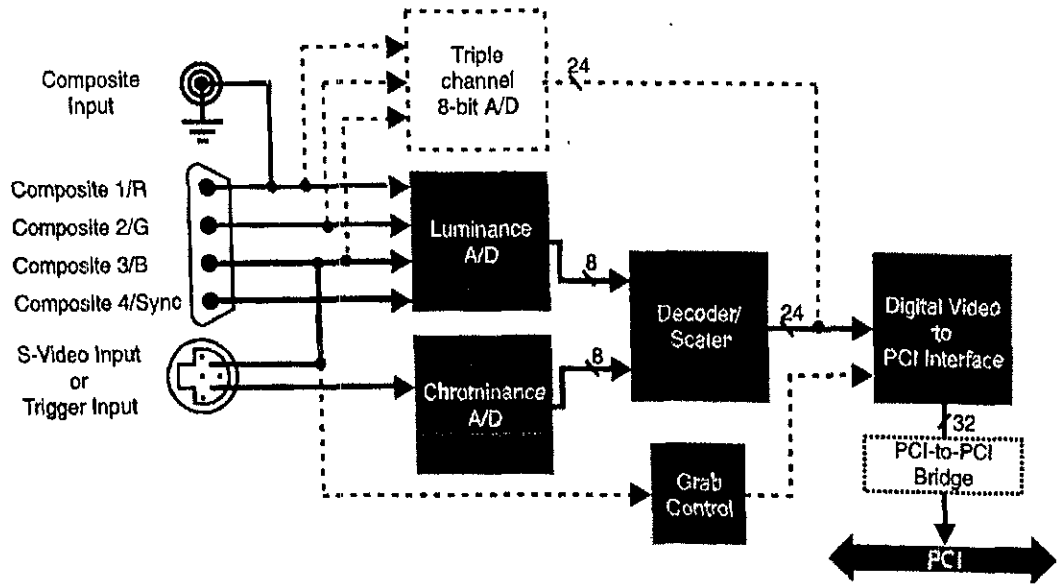


Fig. 3.2. Diagrama a bloques del Frame Grabber Matrox Meteor.

3.1.2.2 Transferencia de datos.

El Frame Grabber con interfaz para bus de 32-bit PCI, soporta transferencias hasta 132 Mbytes/segundo. Provee transferencia de imágenes hacia:

- El sistema de memoria de la PC para procesamiento.
- La memoria de video para despliegue.

a) Velocidades de transferencia.

Este Frame Grabber puede capturar y transferir cuadros (imágenes) a resolución completa como se muestra en la Tabla 3.2. No obstante, las velocidades de transferencia dependen en gran medida de las capacidades del

sistema (PC utilizada). Teóricamente la transferencia de datos para PCI es de 132 Mbytes/segundo, aunque el desempeño actual es mucho menor y, depende de la calidad del sistema para la interfaz del bus PCI.

Por ejemplo, sistemas de bus PCI con un chipset Intel 430NX (Neptune), no ofrecen suficiente ancho de banda para transferencias en tiempo real. Sistemas de bus PCI recientes usan un chipset más nuevo; así el Intel 430FX (Triton) ofrece suficiente ancho de banda para transferencias en tiempo real usando Matrox Meteor.

Tabla 3.2

	NTSC	PAL
Tamaño.	640 × 480.	768 × 576.
Bits por pixel.	8, 15 o 32 bits.	8, 15 o 32 bits.
Numero de cuadros.	30 cuadros/segundo.	25 cuadros/segundo.
Velocidad de transferencia.	Hasta 35 Mbytes/segundo.	Hasta 42 Mbytes/segundo.

Tabla 3.2. Capacidades de transferencia para Matrox Meteor.

b) Transferencias para despliegue.

Las transferencias de datos desde Matrox Meteor hacia la memoria de despliegue puede ser efectuada de las siguientes dos maneras:

- Transferencia directa hacia la memoria de despliegue.

En este método, la transferencia desde Matrox Meteor hacia la memoria de despliegue se hace directamente sin involucrar el CPU de la computadora; sin embargo, requiere el uso de una tarjeta de video especifica de la misma compañía Matrox. Este mecanismo se ilustra en la Fig. 3.3.

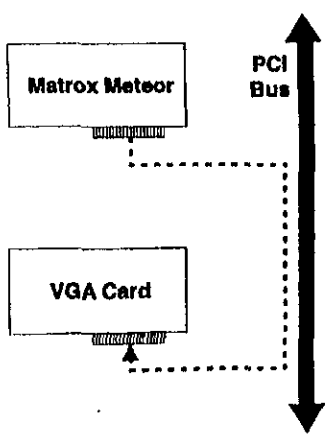


Fig. 3.3. Transferencia directa desde Matrox Meteor hacia la memoria de despliegue.

- Transferencia indirecta hacia la memoria de despliegue.

Este tipo de transferencia ocurre cuando no se dispone de una tarjeta de video que soporte la transferencia directa. Aquí se emplean buffers temporales en el sistema de memoria RAM. Mientras Matrox Meteor está grabando un Frame en un buffer temporal, el driver de despliegue está transfiriendo el Frame anterior al sistema de memoria de despliegue. La ventaja es que cualquier tarjeta de video puede ser utilizada. La transferencia indirecta será en tiempo real si la transferencia desde el sistema RAM del CPU hacia la tarjeta de video es suficientemente rápida. Este mecanismo se ilustra en la Fig. 3.4.

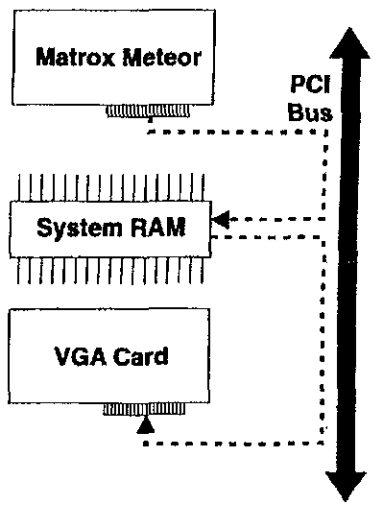


Fig. 3.4. Transferencia indirecta desde Matrox Meteor hacia la memoria de despliegue.

3.1.2.3 Software

El Frame Grabber Matrox Meteor se puede programar usando la librería MIL-Litle, desarrollada en lenguaje "C" e incluida con la tarjeta. MIL-Litle posee comandos para adquisición, manipulación de datos, gráficos y controles de despliegue. Es una librería modular e independiente del hardware, puede manipular imágenes en escala de grises y color. Aplicaciones basadas en MIL-Litle pueden correr sobre cualquier tarjeta VESA compatible, bajo los ambientes DOS extenders, Windows 3.1, Windows 95 y Windows NT.

MIL-Litle maneja objetos físicos (sistemas, digitalizadores, despliegues y buffers de datos) como objetos virtuales. Estos deben ser alojados antes de ser manipulados y destruidos cuando se salga de la aplicación. Para aplicaciones sencillas, estos objetos son alojados por omisión, siendo posible iniciar una aplicación de manera muy sencilla.

MIL-Litle puede grabar pixeles usando diverso número de bits, adquirir imágenes color a 8 o 16 bits, desplegar a 1, 8 o 16 bits en escala de gris o color. La librería incluye funciones para modificar las imágenes, tal como agregar texto o dibujar líneas, círculos etc.

a) Construyendo una aplicación.

Cualquiera que sea la aplicación, se deben efectuar los siguientes pasos:

- Alojar la aplicación. Crea un ambiente de control y ejecución para la aplicación.
- Alojar el sistema. Abre canales de comunicación e inicializa el sistema. Una vez que se ha establecido comunicación entre el Host y el sistema, pueden ser alojados recursos de memoria, de despliegue y entrada de datos. Un sistema de la aplicación puede tener muchos buffers, despliegues y digitalizadores. Esto se ilustra en la Fig. 3.5.

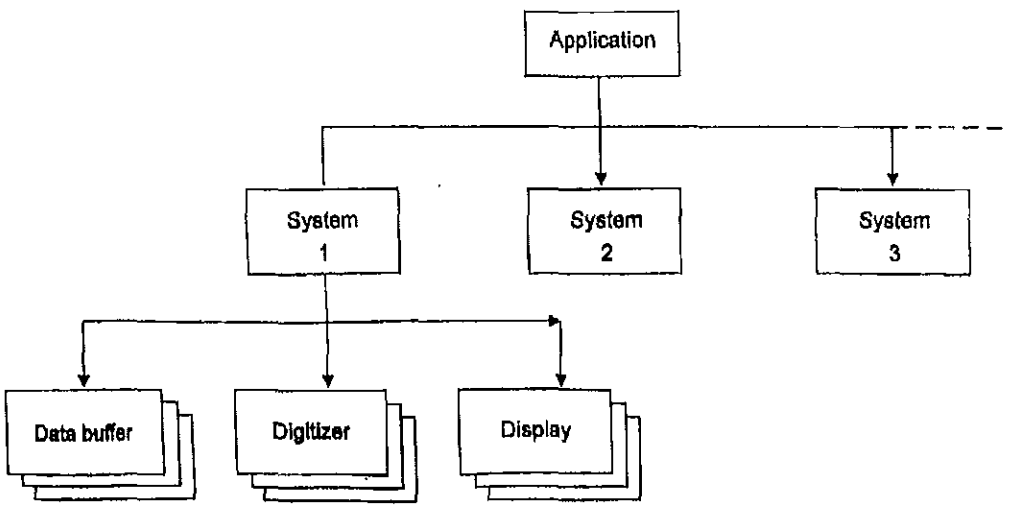


Fig. 3.5. Componentes de una aplicación.

b) Compilando una aplicación.

Para compilar cualquier programa basado en MIL-Litle, se debe incluir la cabecera de la librería *"mil.h"*. Después, se requiere ligar con las librerías apropiadas para el sistema operativo utilizando. Para la versión MIL-Litle de 16 bits se debe utilizar Borland C 3.1 o superior, mientras en la versión MIL-Litle de 32 bits, Borland C 4.5 o superior.

Para mayores detalles sobre las características del Frame Grabber y librería MIL-Litle, se recomienda consultar los manuales de las referencias [17], [18] y [19].

3.2 Estructura Mecánica.

Se requiere la construcción de una estructura mecánica que proporcione movimiento a la cámara de video CCD, con el objetivo de efectuar las acciones de control calculadas por el CLB y, ejecutadas por una electrónica de control de motores que es descrita en la siguiente sección.

El diseño de esta estructura mecánica requiere considerar los siguientes aspectos importantes:

- Peso y dimensiones de la cámara de video CCD. Influye en la determinación del tamaño de la estructura mecánica así como de los motores a utilizar. El peso de la cámara es aproximadamente 40 gramos, más su cubierta en acrílico de 70 gramos. Las medidas aproximadas son 5cm×5cm×3cm.

- Peso y dimensiones de los motores a utilizar. Por razones de costos se seleccionan motores de pasos comerciales que se ajustan lo más posible a los requisitos que se señalan en la siguiente sección para el diseño de la electrónica. Su peso aproximado es de 100 gramos y sus dimensiones son 3cm×3cm×3cm. Por tal razón, el diseño de la estructura mecánica debe tomar en cuenta esta selección.

Por otra parte, la estructura mecánica debe cumplir con las siguientes especificaciones de diseño:

- Movimientos de la cámara en coordenadas esféricas. Esta consideración va a permitir cubrir un espacio esférico mediante un movimiento en dos ángulos y, en consecuencia se requieren sólo dos motores.

- Capacidad para movimientos precisos y menores al paso del motor. Esta consideración requiere la utilización de reductores de velocidad para cada motor de pasos. Este reductor es importante, en primer lugar, porque permitir pasos menores a 1.8 grados (típico de los motores comerciales), en segundo lugar, aumenta el torque de los motores y, en tercer lugar, permite que la única manera de desplazar la cámara sea por medio de los motores, en consecuencia minimiza los efectos de inercia debido tanto al peso de la cámara como de la misma estructura. Por el requisito de precisión, los reductores deben tener un pequeño backlash a la salida. La relación de los reductores quedará establecida en función del peso de la cámara, estructura mecánica y motores de pasos.

La estructura mecánica está actualmente en proceso de diseño y construcción con la asesoría del departamento de Diseño Mecánico del Centro de

Instrumentos. Específicamente se están haciendo pruebas para la elección del reductor.

3.3 Electrónica de Control de Motores.

3.3.1 Requisitos.

La adquisición y procesamiento de las imágenes adquiridas, así como el control para el seguimiento del objeto, consumen muchos recursos de cómputo. Esta es la razón para diseñar una electrónica que reciba y ejecute las correcciones calculadas por el programa de procesamiento, sin consumir más tiempo de cómputo; de esta forma la PC es completamente liberada del control de motores.

3.3.2 Selección de motores.

La cámara CCD utilizada es pequeña, con un peso aproximado de 40 gramos más su cubierta de 70 gramos, de forma tal que la estructura mecánica y tamaño de los motores se determinan conjuntamente.

Se eligieron motores de pasos comerciales por dos razones:

- Facilidad para efectuar su control (existen drivers comerciales).
- No se requiere retroalimentación para controlar posición.

Las características principales de los motores de pasos elegidos debieron cumplir con:

- *Dimensiones no mayores a las de la cámara CCD.* Esta consideración permite minimizar el tamaño y peso de la estructura mecánica, así como el esfuerzo de uno de los motores para desplazar a su homólogo.

- *Pasos no mayores a dos grados.* Garantiza pequeños desplazamientos de la estructura mecánica para relaciones de engranes superiores a 5:1.

- *Velocidades no menores a 1 Khz.* Asegura correcciones pico (180 grados) no mayores a 1 segundo, de forma que una corrección promedio de 18 grados no sea mayor a 1/10 de segundo y, por consiguiente, se mantengan por debajo de la velocidad de procesamiento digital para la extracción de su centroide.

- *Torque considerable.* No es tan crítico por dos razones: primera, una cámara muy ligera, segunda, por la utilización de un reductor de velocidad.

Los motores de pasos elegidos, mostrados en la Fig. 3.6, cumplen con los requisitos anteriores y tiene una resolución de 200 pasos completos y 400 medios pasos, consumo de corriente por cola de 500 mA y peso de 100 gramos.

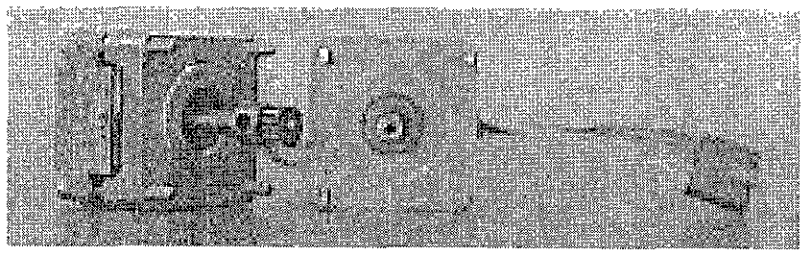


Fig. 3.6. Motores de pasos utilizados.

3.3.3 Diseño de la electrónica.

El requisito de liberar el procesador del control de motores, limita su trabajo a calcular la corrección y comunicársela a la electrónica. Sin embargo, complica el diseño de la electrónica al responsabilizarla completamente de llevar los motores a la nueva posición.

La electrónica debe ser capaz de controlar de forma independiente cada coordenada, así como utilizar al máximo las características del driver comercial para motores de pasos, como son: sentido del desplazamiento y movimiento en paso completo o medio paso.

a) Máxima corrección.

La máxima corrección y la relación en la caja de engranes, determinan completamente el número de pasos máximo por corrección y el tamaño del diseño de la electrónica.

La corrección máxima de 180 grados requiere de 100 pasos completos. Con una caja de relación máxima de 20:1, se necesitan 2000 pasos completos o 4000 medios pasos para la corrección de 180 grados. De esta manera la electrónica puede desplazar un máximo de 4000 medios pasos (180 grados) por corrección.

b) Elección del driver para motores de pasos.

Se eligió el circuito integrado MC3479P capaz de manejar un motor de pasos de dos devanados. El circuito posee las siguientes características:

- Una sola fuente de alimentación de +7.2 a 16.5 V.
- Corriente por devanado de hasta 350 mA.

- Selección lógica de paso completo o medio paso.
- Selección lógica del sentido de movimiento para el motor.

La conexión típica para aplicaciones como la nuestra se ilustra en la Fig. 3.7 y, en la Fig. 3.8 se ilustra el diagrama a bloques del driver. El valor de la resistencia R_B determina la corriente máxima por devanado, con base en la siguiente expresión:

$$R_B = \frac{V_M - 0.7v}{I_{od} * 0.86}$$

donde "I_{od}" es la corriente máxima deseada por devanado en mA y "V_M" es el voltaje aplicado al pin de alimentación del driver que puede estar entre +7.2V y 16.5V.

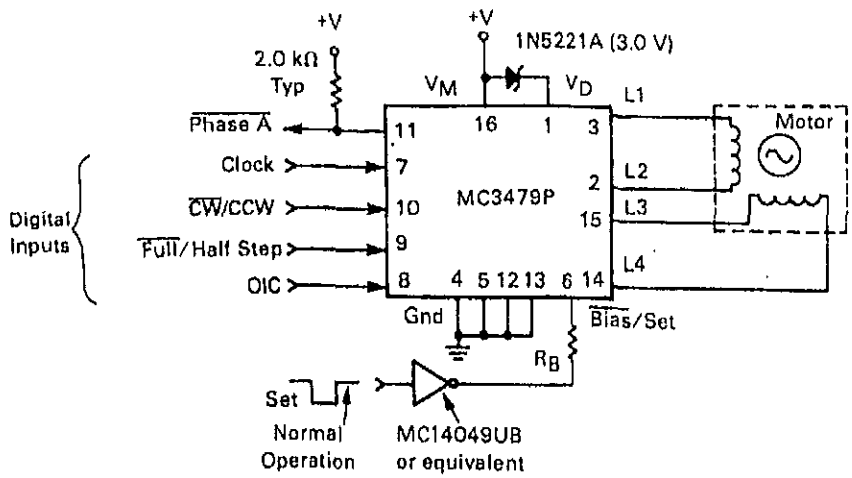


Fig. 3.7. Conexión típica para el driver MC3479P.

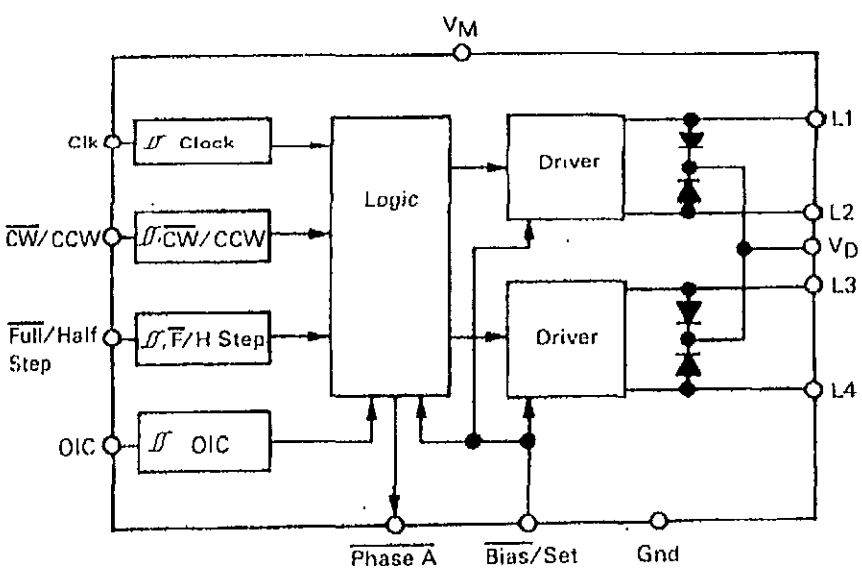


Fig. 3.8. Diagrama a bloques del driver MC3479P.

c) Elección del circuito de potencia.

Los motores de pasos utilizados consumen 500 mA por devanado, de manera que el driver MC3479P no puede proporcionar la corriente necesaria. Por tanto, se eligió el circuito LM18293 que es un driver sencillo para motores de DC o de pasos, pero capaz de manejar hasta 1 A. El diagrama de conexión típico se ilustra en la Fig. 3.9.

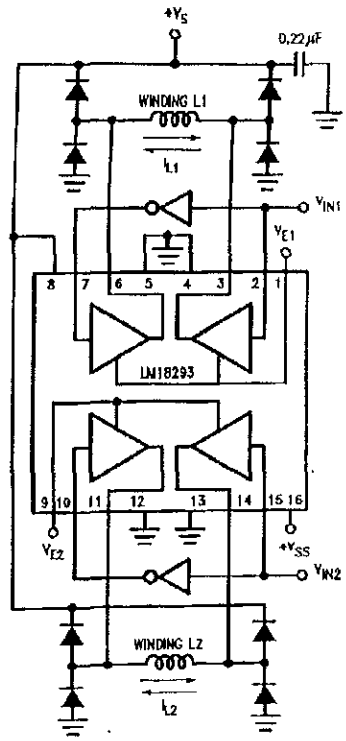


Fig. 3.9. Conexión típica del circuito de potencia.

d) Interfaz entre el driver MC3479P y el circuito de potencia LM18293.

La interface entre ambos circuitos es sencilla, consiste en utilizar el integrado MC3479P como el driver de control. Se establece la corriente de devanados muy por debajo de los 350 mA fijando R_B y V_M a 2.2K y 5.0V respectivamente, de manera que la corriente I_{od} es de 2.3 mA, suficiente para operar el integrado LM18293. Las salidas por devanado del integrado MC3479P se conectan a las entradas del integrado LM18293, suprimiendo los inversores de la Fig. 3.9.

e) Interfaz entre la PC y la electrónica de control.

El flujo de información entre la PC y la electrónica de control se efectúa utilizando la interfaz periférica programable 8255A -ilustrada en la Fig. 3.10- la cual posee las siguientes características esenciales de utilidad para nuestro diseño:

- Tres puertos programables de ocho bits cada uno.
- Completamente TTL.

Posee tres puertos paralelos para intercambio de información denominados puertos A, B y C, los cuales se dividen en dos grupos: grupo A contiene los puertos A y la parte alta del puerto C; grupo B contiene los puertos B y la parte baja del puerto C. También posee un puerto de control para definir las características de entrada o salida de cada puerto de datos.

Posee tres modos de operación, utilizamos el modo básico de entrada/salida, que permite usar la interfaz como tres puertos paralelos independientes.

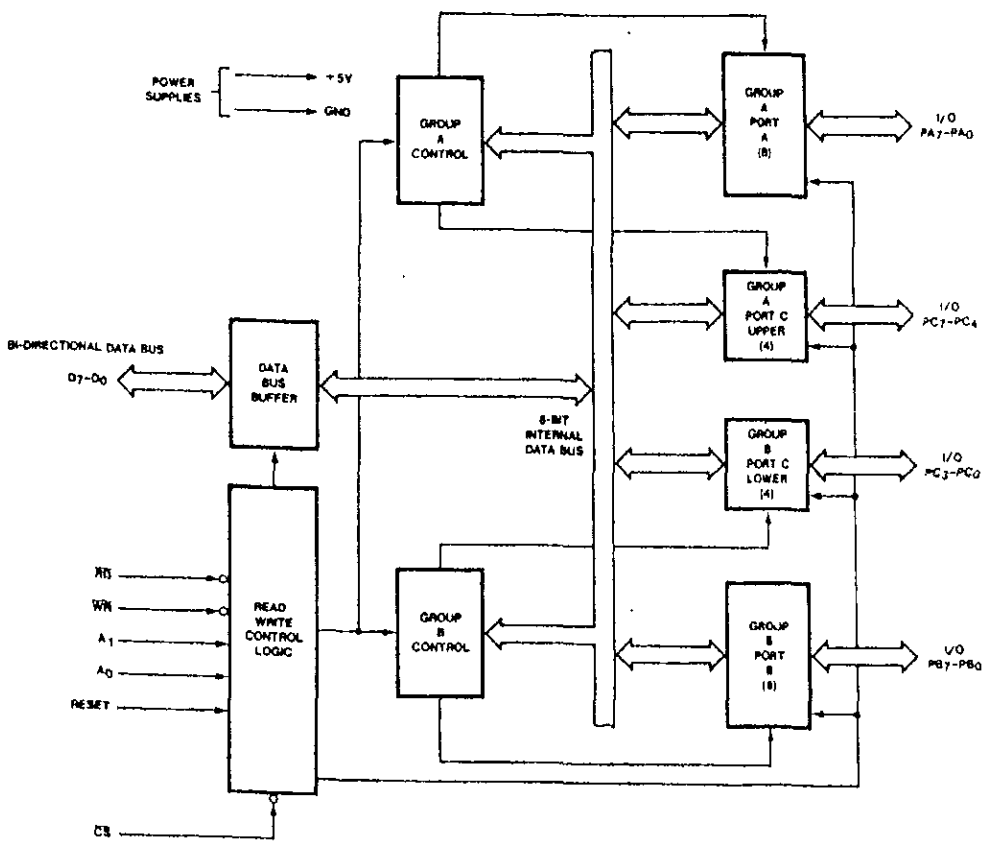


Fig. 3.10. Diagrama a bloques de la interfaz periférica 8255A.

El modo de operación del grupo A y B de la interfaz se define en los bits 5, 6 y 2 del puerto de control respectivamente; para nuestros requerimientos la combinación 00 selecciona el modo básico de entrada/salida en el grupo A y la combinación 0 en el grupo B. Los bits 0, 1, 3 y 4 determinan la definición de los puertos como se ilustra en la Tabla 3.3. El puerto C tiene la característica de poder emplearse como dos puertos separados cada uno de 4 bits.

Tabla 3.3

Bit 4	Bit 3	Bit 1	Bit 0	Puerto A	Puerto C (alta)	Puerto B	Puerto C (baja)
0	0	0	0	Salida	Salida	Salida	Salida
0	0	0	1	Salida	Salida	Salida	Entrada
0	0	1	0	Salida	Salida	Entrada	Salida
0	0	1	1	Salida	Salida	Entrada	Entrada
0	1	0	0	Salida	Entrada	Salida	Salida
0	1	0	1	Salida	Entrada	Salida	Entrada
0	1	1	0	Salida	Entrada	Entrada	Salida
0	1	1	1	Salida	Entrada	Entrada	Entrada
1	0	0	0	Entrada	Salida	Salida	Salida
1	0	0	1	Entrada	Salida	Salida	Entrada
1	0	1	0	Entrada	Salida	Entrada	Salida
1	0	1	1	Entrada	Salida	Entrada	Entrada
1	1	0	0	Entrada	Entrada	Salida	Salida
1	1	0	1	Entrada	Entrada	Salida	Entrada
1	1	1	0	Entrada	Entrada	Entrada	Salida
1	1	1	1	Entrada	Entrada	Entrada	Entrada

Tabla 3.3. Definición de los puertos para la interfaz 8255.

f) Funciones de la PC y electrónica de control para el movimiento de motores

La PC tiene por tareas:

- Enviar el número de pasos que para cada una de las coordenadas deben desplazarse los motores.
- Enviar señales para controlar tanto el sentido como tamaño de paso.
- Enviar señales de inicio y reset.

La electrónica de control debe:

- Desplazar los motores el número de pasos en el sentido indicado por la PC.
- Completado el movimiento de los motores, estos deben permanecer en paro total y consumo mínimo de potencia.
- Habilitar una señal que permita saber a la PC que la acción ha sido completada por la electrónica, con el objetivo de evitar un conflicto entre la PC y la electrónica de control.

El número de pasos a desplazar cada motor se almacena en contadores con capacidad de carga paralela. Después, se hace un conteo descendente hasta llegar a la cuenta cero. Anteriormente se calculó la máxima corrección que consiste hasta de 2000 pasos o 4000 medios pasos, de forma tal que se necesitan contadores de 12 bits; sin embargo, comercialmente sólo existen contadores de 4 bits con características de carga paralela y señal de acarreo. Se emplearon contadores 74ls191 de 4 bits que pueden ser conectados en cascada para permitir formar un contador con mayor resolución.

g) Definición de los puertos del 8255 y asignación de funcionamiento.

Las tareas mencionadas en el inciso anterior para la PC y electrónica de control, definen todos los puertos como de salida excepto la parte baja del puerto C, que se define como entrada; una línea de este puerto permite a la PC saber cuando se ha completado la acción de control.

El puerto A se define como de datos y, los puertos B y C (parte alta) como señales de control. La asignación por pines se ilustra en las Tablas 3.4, 3.5 y 3.6, se anexa el símbolo utilizado y significado de cada bit.

h) Aseguramiento de paro en los motores.

Los motores son habilitados o detenidos usando un flip-flop j-k (7473) en configuración T, de manera que la señal de inicio enviada por la PC permite iniciar el movimiento y, la señal de acarreo del último contador posibilita terminar el movimiento de los motores. No obstante, el contador 74ls191 sólo envía un pulso de fin de cuenta, de manera que en estado de paro, el flip-flop es muy sensible a perturbaciones electrostáticas dado que se dispara por flanco. En consecuencia, requiere un circuito adicional que utilice el pulso de acarreo del último contador para asegurar el paro de los motores mediante la limpieza continua del flip-flop. Esto se logra mediante un circuito que se dispara en oscilación con el pulso de acarreo del contador y se detiene con una señal de la PC. Este circuito es un oscilador construido con circuitos 74ls221.

i) Diagrama esquemático del circuito global.

El esquemático del circuito completo está dividido en tres secciones principales:

1) Sección de conteo de número de pasos. Este esquemático se ilustra en la Fig. 3.11. Está constituido por seis contadores 74ls191 divididos en dos grupos, el primero para el conteo de pasos en la coordenada X, el siguiente para la coordenada Y. En cada grupo los contadores están etiquetados como *bajo, medio y alto*. Están conectados en cascada haciendo uso de la señal de acarreo como

reloj para el siguiente contador. Todos los contadores cuentan en forma descendente.

Tabla 3.4

Puerto A	Símbolo	Significado
A0	A0	Pasos dato bit 0
A1	A1	Pasos dato bit 1
A2	A2	Pasos dato bit 2
A3	A3	Pasos dato bit 3
A4	A4	Pasos dato bit 4
A5	A5	Pasos dato bit 5
A6	A6	Pasos dato bit 6
A7	A7	Pasos dato bit 7

Tabla 3.4. Asignación de pines para el puerto A.

Tabla 3.5

Puerto B	Símbolo	Significado
B0	X_H/~F	Coordenada X: H=medio paso (1) , F=paso completo (0).
B1	Y_H/~F	Coordenada Y: H=medio paso (1) , F=paso completo (0).
B2	X_D/~I	Coordenada X: D=sentido horario (1) , I=sentido antihorario (0).
B3	Y_D/~I	Coordenada Y: D=sentido horario (1) , I=sentido antihorario (0).
B4	L_X_BM	Carga partes baja y media de contadores X. Habilita con (0).
B5	L_X_A	Carga partes alta de contadores X. Habilita con (0).
B6	L_Y_BM	Carga partes baja y media de contadores Y. Habilita con (0).
B7	L_Y_A	Carga partes alta de contadores Y. Habilita con (0).

Tabla 3.5. Asignación de pines para el puerto B.

Tabla 3.6

Puerto C	Símbolo	Significado
C0	Fin	Habilita a (1) para indicar a la PC que terminó el movimiento.
C1	----------	No usado.
C2	----------	No usado.
C3	----------	No usado.
C4	Start_X	Indica a la electrónica que inicie movimiento en X (↓)
C5	Start_Y	Indica a la electrónica que inicie movimiento en Y (↓)
C6	Para_MX	Detiene el oscilador de aseguramiento de paro en X (1)
C7	Para_MY	Detiene el oscilador de aseguramiento de paro en Y (1)

Tabla 3.6. Asignación de pines para el puerto C.

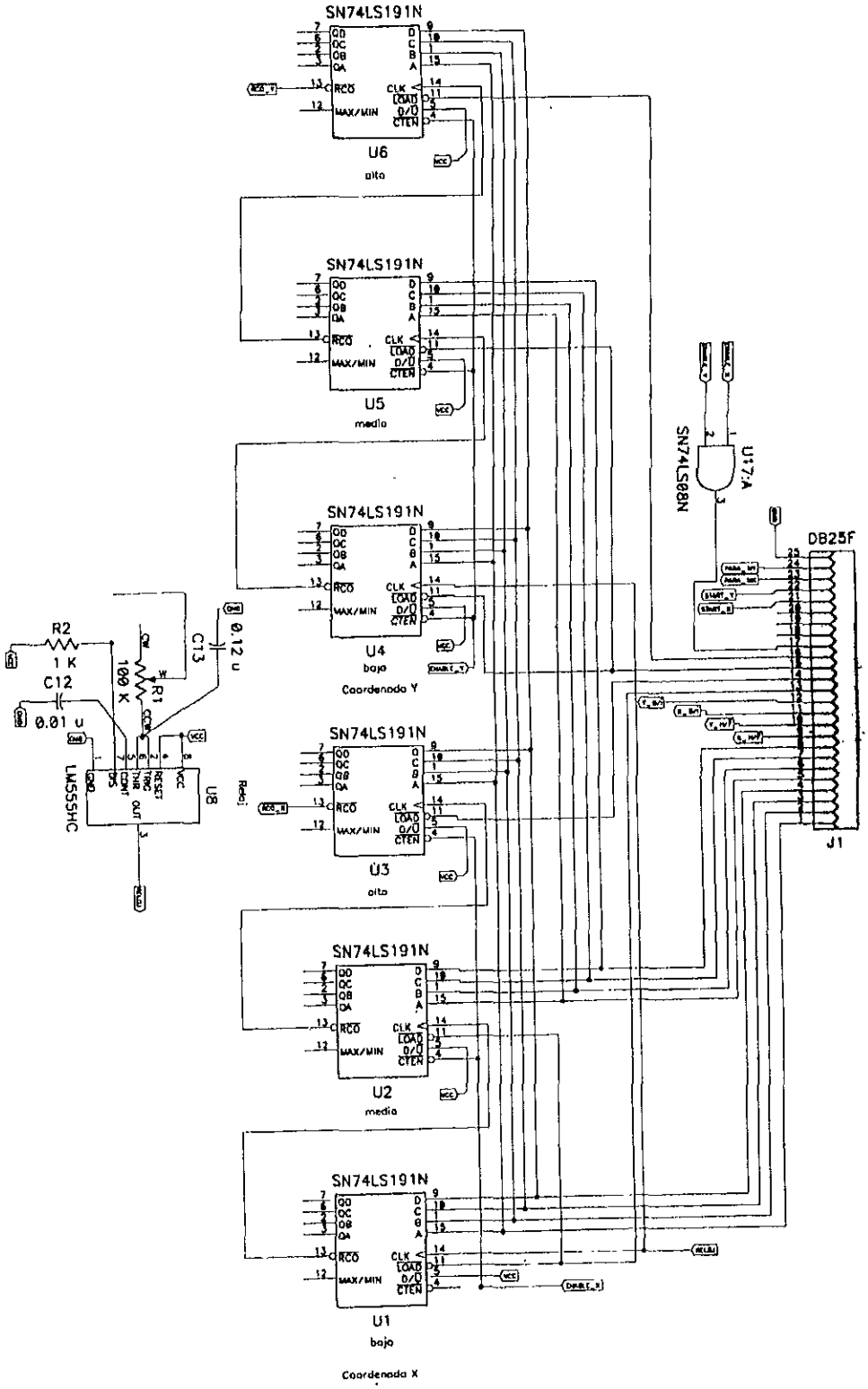


Fig. 3.11. Esquemático para el conteo de pasos.

La señal de acarreo del contador, con la parte más alta de la cuenta, es usada en las siguientes secciones para determinar si la cuenta ha finalizado y detener los motores. Esta señal de acarreo presenta un pulso bajo de duración igual a la parte baja del reloj, se simbolizan para la coordenada X como *RCO_X* y para la coordenada Y como *RCO_Y*.

Las partes *bajas* y *medias* de los contadores son cargadas de una sola vez con los datos provenientes de la PC en el puerto de datos (puerto A), las partes *altas* de los contadores son cargadas por una segunda escritura del puerto de datos, escribiendo sólo los cuatro bits menos significativos. Las señales de control para la carga son respectivamente *L_X_BM*, *L_X_A*, *L_Y_BM*, *L_Y_A*. De esta forma, se requiere de 4 escrituras al puerto de datos y al puerto de control para hacer la carga de los contadores de ambas coordenadas.

2) Sección de aseguramiento del paro de motores. Se emplea como elemento esencial el circuito monostable 74ls221, el diagrama lógico y su tabla de verdad se ilustran en la Fig. 3.12. Cada chip contiene dos monostables e interconectándolos es posible generar un oscilador que se habilite por disparo. El esquemático del circuito oscilador para ambas coordenadas se ilustra en la Fig. 3.13. El circuito es idéntico para cada coordenada, de manera que sólo se describe el funcionamiento para la coordenada X.

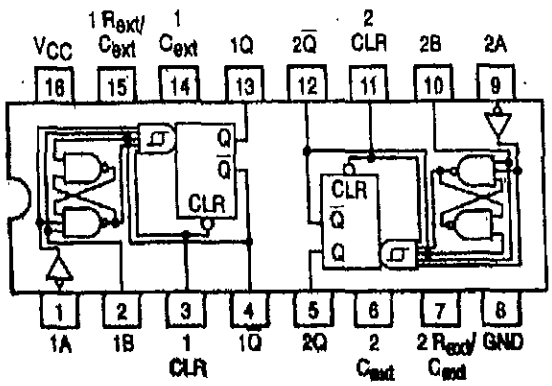
Recibe las entradas *RCO_X* y *Para_MX*; la primera, es un pulso bajo generado por el acarreo del contador que determina la parte alta de la cuenta, este pulso indica que se ha finalizado la cuenta. La segunda, es enviada desde la PC y se utiliza para detener el circuito oscilador.

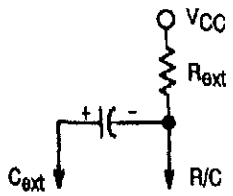
El flanco de bajada del pulso *RCO_X* dispara un monostable que reduce el ancho de *RCO_X* a 10 microsegundos aproximadamente, el flanco de bajada de este nuevo pulso es usado para disparar por primera vez el oscilador conformado por los siguientes tres monostables; la salida del último se retroalimenta a través de la compuerta OR y AND para continuar en oscilación con un periodo de aproximadamente 100 microsegundos.

Un nivel lógico 0 en la señal *Para_MX* permite que el oscilador siga trabajando, mientras un nivel lógico de 1 por un periodo mayor a 100 microsegundos detiene el oscilador y lo deja preparado para ser disparado por otro pulso bajo en RCO_X.

La señal de oscilación es tomada del segundo monostable y enviada al pin de limpieza del flip-flop a través de una compuerta AND, permitiendo así mantener detenidos los motores, aún en un ambiente con alta electrostática. Las dos señales de salida de los flip-flop son *ACABE_X* y *ENABLE_X*; la primera, es enviada a la sección de potencia para controlar el estado del reloj que se alimenta al driver del motor de pasos; la segunda, se encarga de detener los contadores e

indicar a la PC que se ha terminado el control. Las señales para la coordenada Y efectúan exactamente lo mismo.





FUNCTION TABLE (EACH MONOSTABLE)

INPUTS			OUTPUTS	
CLEAR	A	B	Q	$\overline{Q}$
L	X	X	L	H
X	H	X	L	H
X	X	L	L	H
H	L	↑	⎍	⎎
H	↓	H	⎍	⎎
↑	L	H	⎍	⎎

Fig. 3.12. Circuito monostable 74ls221.

3) Sección de potencia para la alimentación de los motores. Utiliza los circuitos MC3479P y el LM18293, el primero es el driver que controla el motor de pasos y, el segundo, permite dar la potencia necesaria. El circuito es simétrico para ambas coordenadas de manera que se describe sólo para la coordenada X.

El MC3479P ejecuta un paso por cada ciclo de reloj recibido, esta sección recibe una señal de reloj filtrada con una compuerta AND según el estado de la señal *ACABE_X*, de manera que si la cuenta ha terminado esta señal está en cero lógico y, en consecuencia la señal de reloj es deshabilitada impidiendo el funcionamiento del motor. También recibe señales *X_D/I* y *X_H/F* que controlan el sentido de giro y tamaño del paso. El esquemático se ilustra en la Fig. 3.14.

Existen dos señales de reloj, la primera denominada *RELOJ* y la segunda *RELOJ_R*, ilustradas en las Figs. 3.11 y 3.14 respectivamente. La señal *RELOJ* se emplea para controlar la cuenta de los contadores y, la señal *RELOJ_R* para controlar el driver MC3479P. La señal *RELOJ* de aproximadamente 1 Khz se genera con un circuito integrado 555, mientras que la señal *RELOJ_R* es la señal *RELOJ* retrasada 100 microsegundos, este retraso se requiere para evitar que la señal de *RELOJ* esté presente en la compuerta AND antes que la señal de control proporcionada por la terminal *ACABE_X*; es decir, evita que siempre se de un paso no controlado. Si esta señal *RELOJ_R* no se utilizara sería imposible hacer movimientos de cero pasos.

El LM18293 recibe la señal *ACABE_X*; la cual, deshabilita la alimentación de los motores cuando la cuenta ha terminado, permitiendo disminuir a cero el consumo de potencia en los motores de pasos.

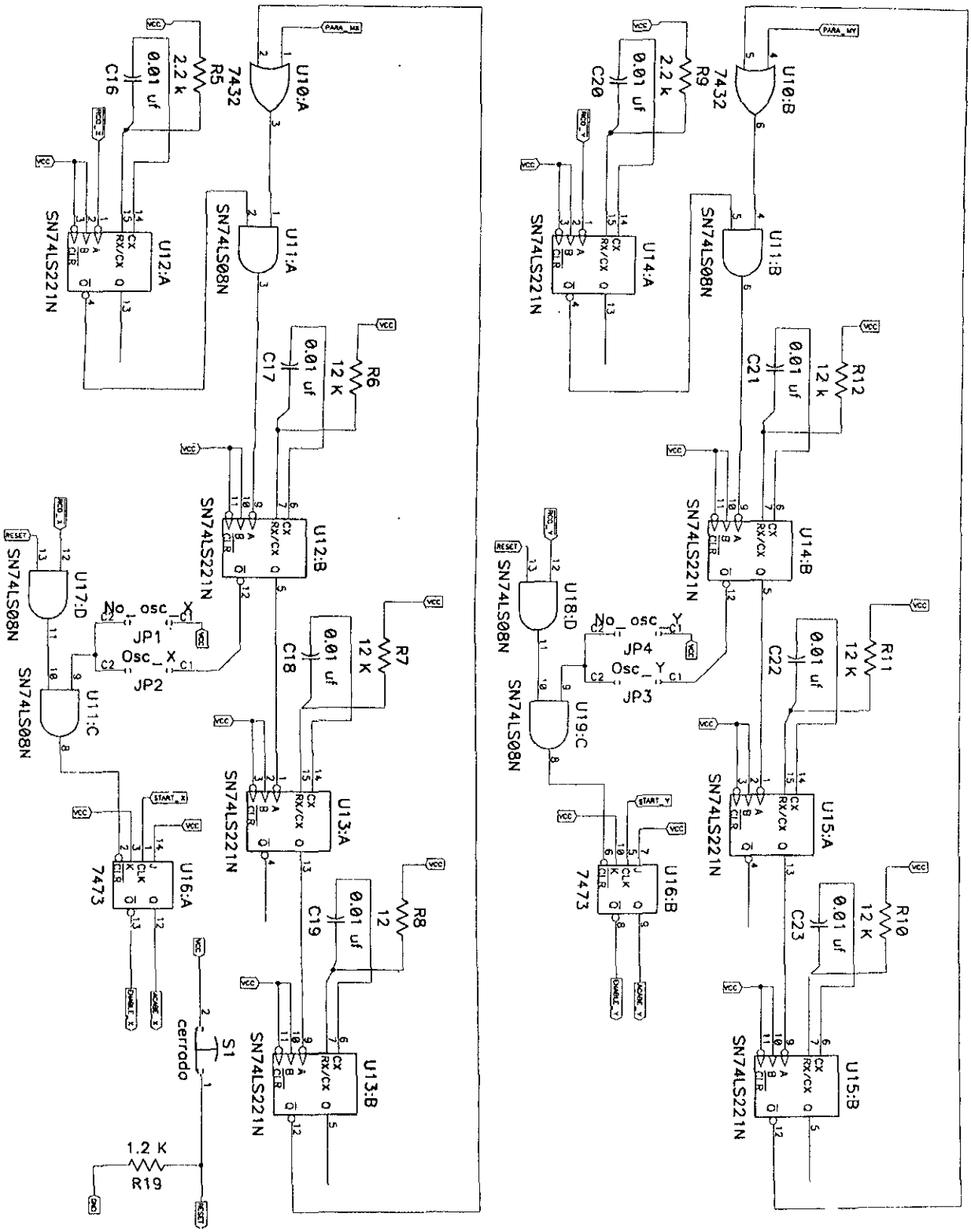


Fig. 3.13. Esquemático para el aseguramiento de paro de motores.

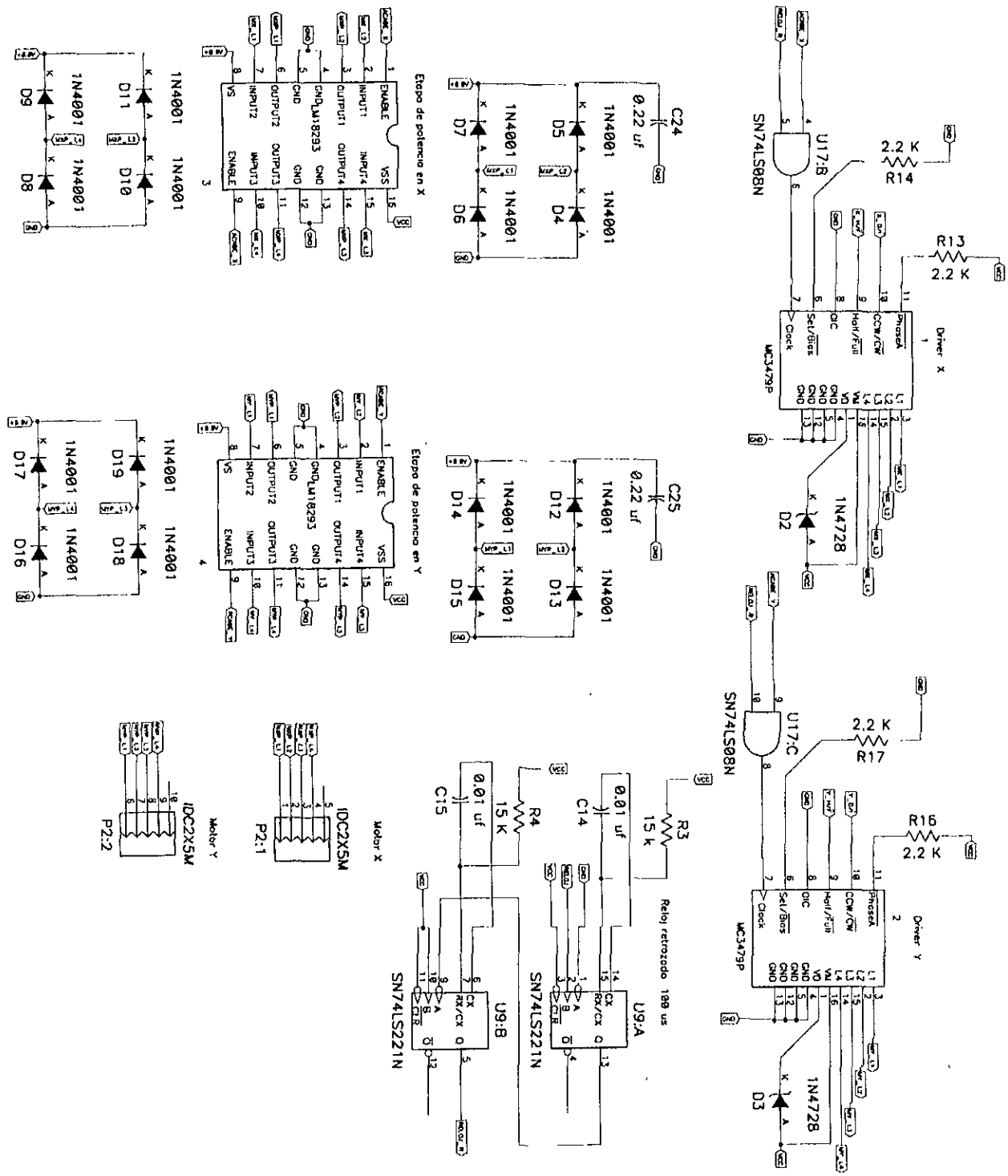


Fig. 3.14. Esquemático para la sección de potencia.

3.3.4 Diseño de circuitos impresos.

Los impresos para los circuitos de las Figs. 3.11, 3.13 y 3.14 se diseñaron en Tango PRO PCB según las referencias [20], [21], [22], [23] y [24]. Todos están construidos en dos caras, de manera que las ilustraciones muestran la caras por separado, además de la cara de componentes. Se crearon las siguientes tarjetas:

1) Tarjeta de control de la lógica. Contiene los contadores para cada una de las coordenadas y el flip-flop para el estado de los motores. Se ilustran en las Figs. 3.15, 3.16 y 3.17 las caras superior, inferior y de componentes.

2) Tarjeta de aseguramiento de paro de motores. Comprende los monostables para generar la oscilación que asegura el paro de motores. Se muestra en las Figs. 3.18, 3.19 y 3.20 las caras superior, inferior y de componentes.

3) Tarjeta de potencia. Abarca el driver y el circuito de potencia. Se ejemplifica en las Figs. 3.21, 3.22 y 3.23 las caras superior, inferior y de componentes.

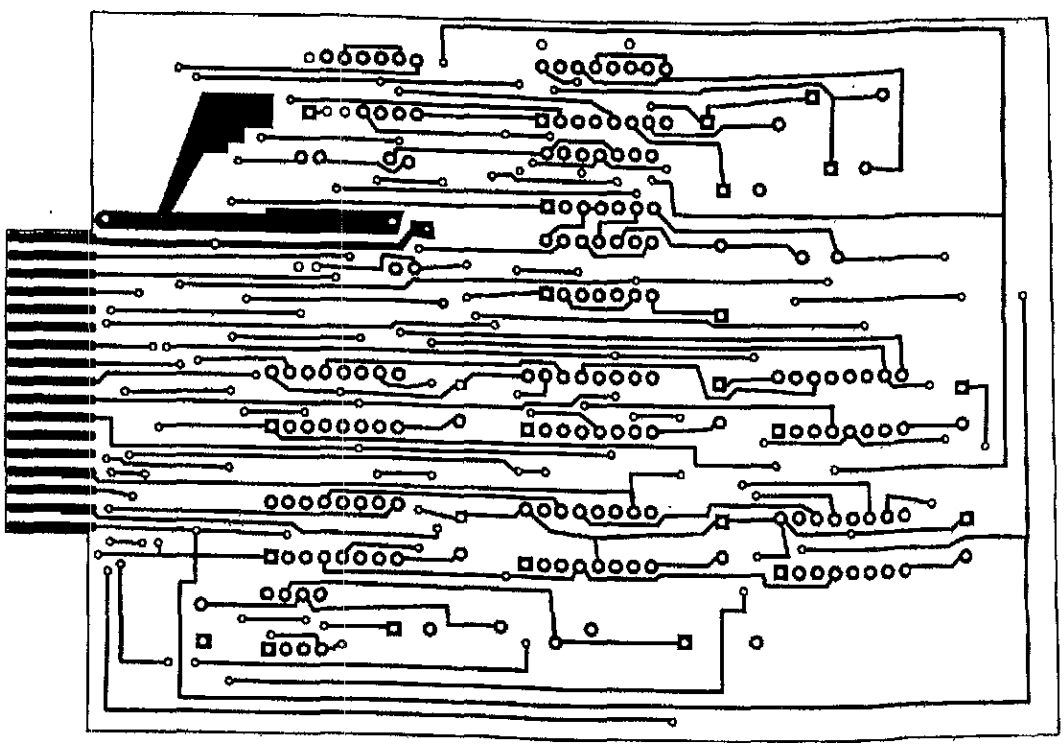


Fig. 3.15. Impreso para el conteo de pasos. Cara superior.

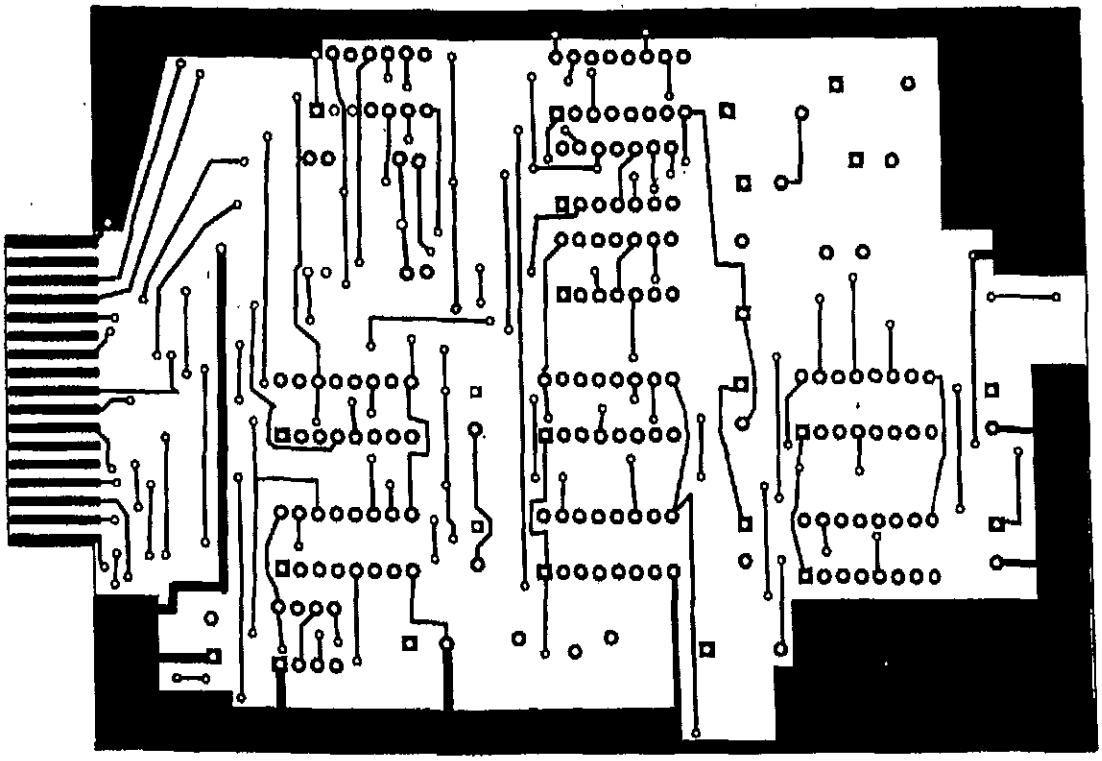


Fig. 3.16. Impreso para el conteo de pasos. Cara inferior.

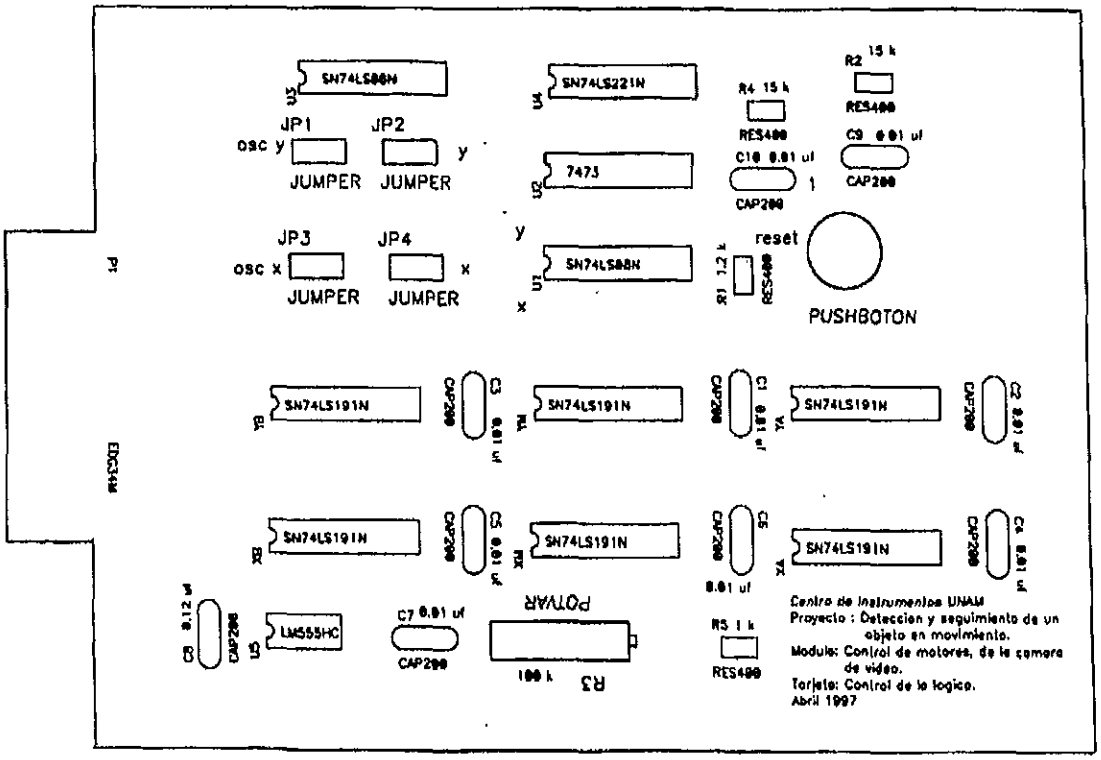


Fig. 3.17. Impreso para el conteo de pasos. Cara de componentes.

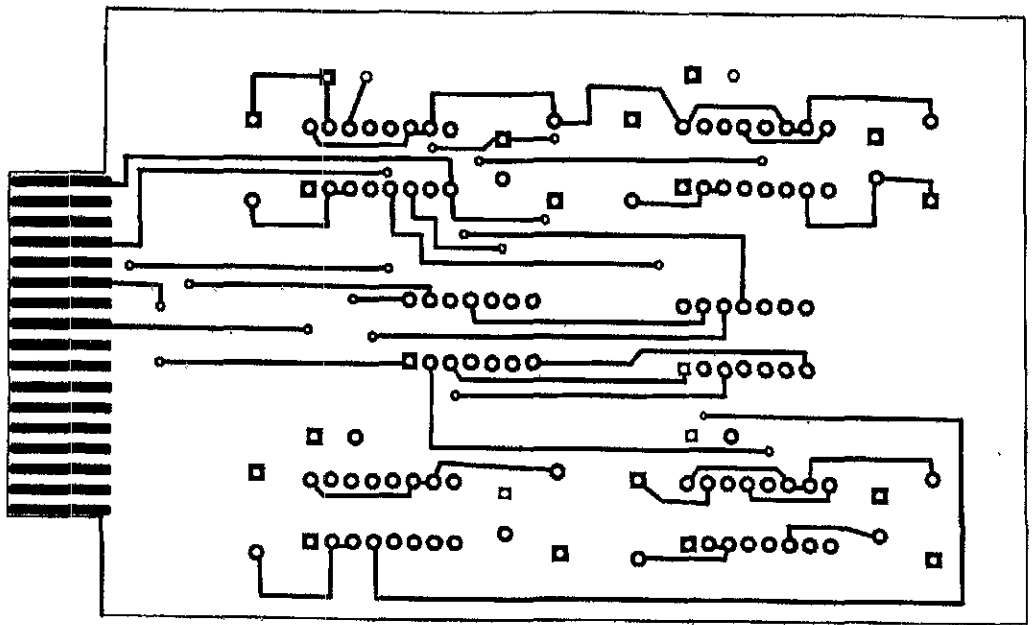


Fig. 3.18. Impreso para el aseguramiento de paro de motores. Cara superior.

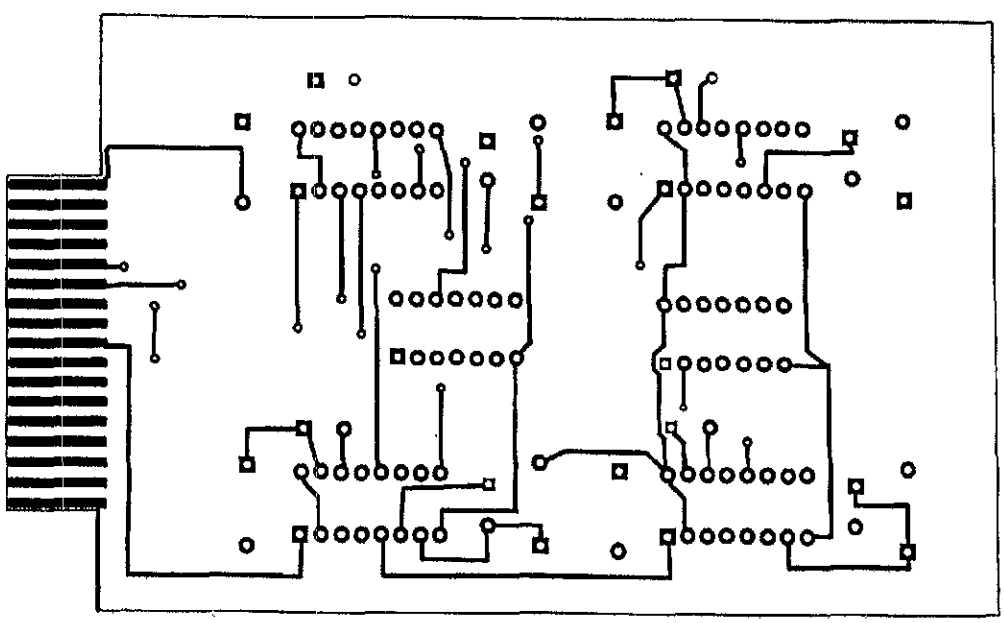


Fig. 3.19. Impreso para el aseguramiento de paro de motores. Cara inferior.

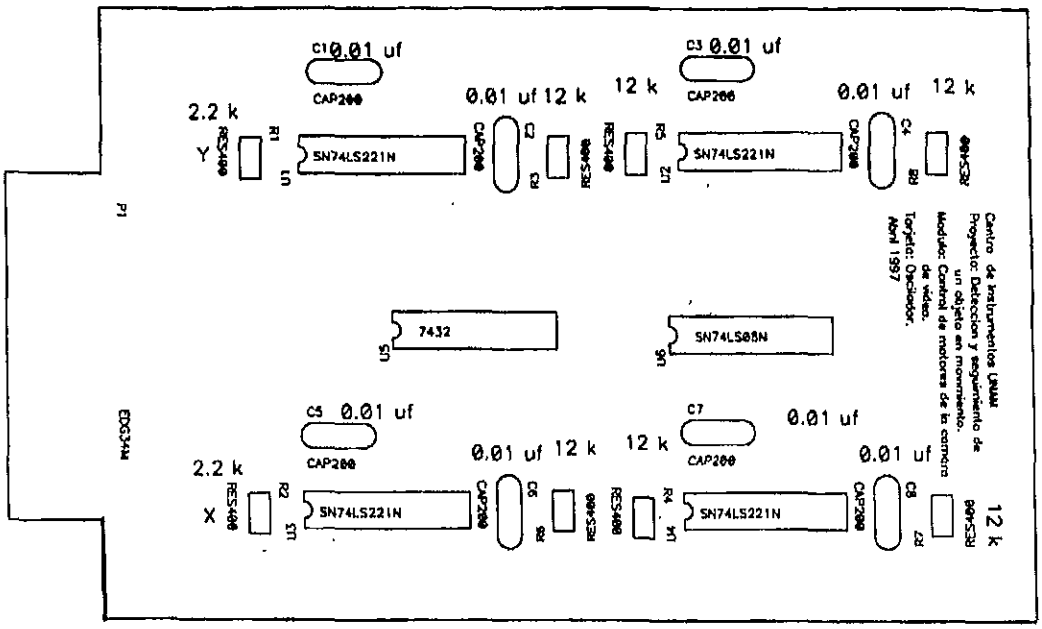


Fig. 3.20. Impreso para el aseguramiento de paro de motores. Cara de componentes.

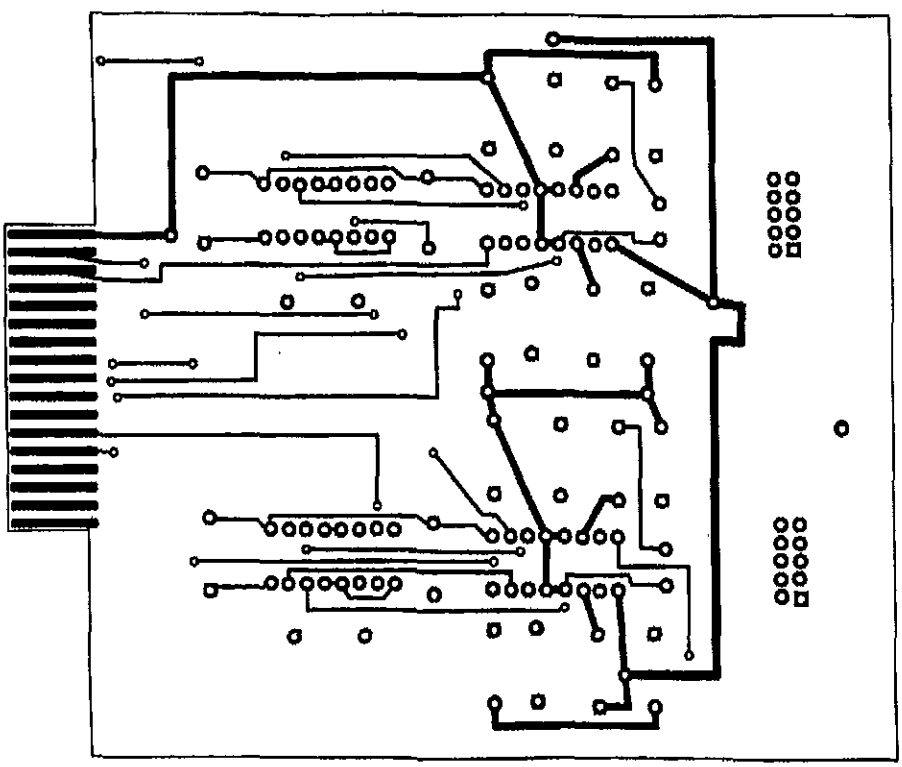


Fig. 3.21. Impreso para la sección de potencia. Cara superior.

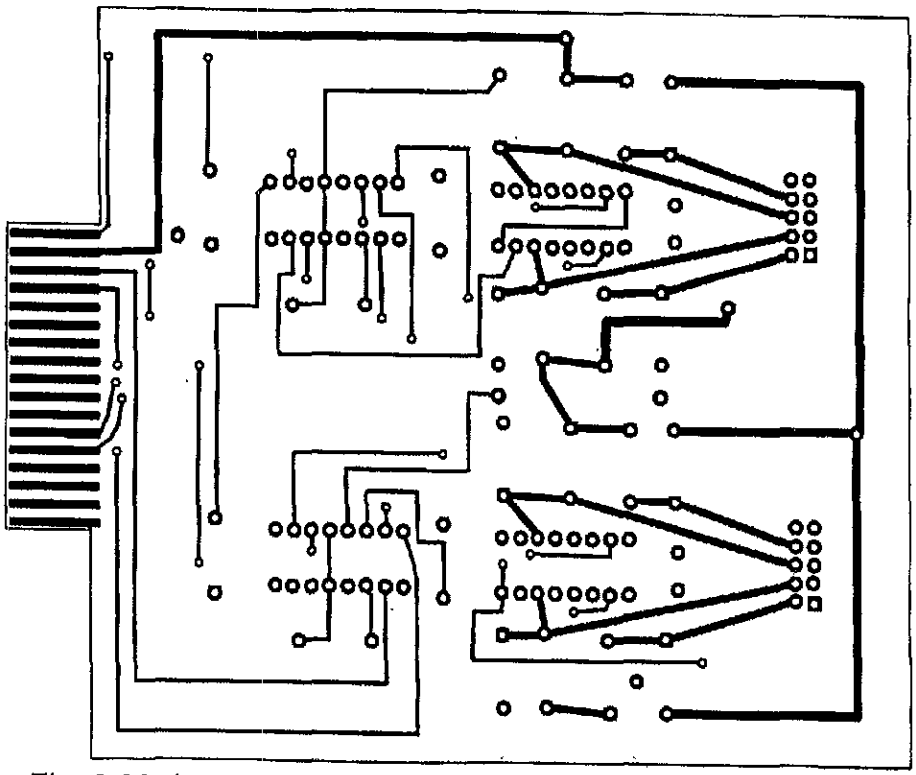


Fig. 3.22. Impreso para la sección de potencia. Cara inferior.

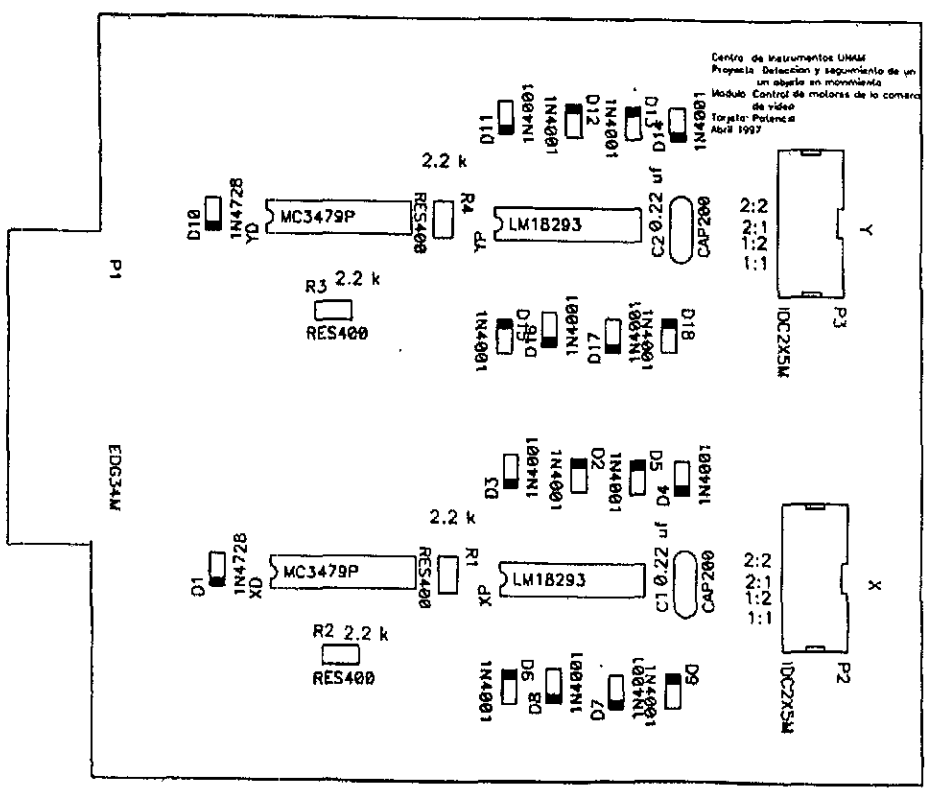


Fig. 3.23. Impreso para la sección de potencia. Cara de componentes.

4) Tarjeta madre. Cada una de las tarjetas anteriores esta diseñada para conectarse en un slot de 34 pines; por lo cual la tarjeta madre contiene 3 slots de 34 pines conectados a un bus común, el cual con un DB25 se conecta a la computadora. El impreso se localiza en las Figs. 3.24, 3.25 y 3.26 para las caras superior, inferior y de componentes. La asignación de pines tanto del DB25 como del SLOT de 34 pines se establece en la tabla 3.7.

Tabla 3.7

No. Pin DB25	No. Pin Slot 34	Descripción o símbolo
1	1	Puerto Datos A0
2	2	Puerto Datos A1
3	3	Puerto Datos A2
4	4	Puerto Datos A3
5	5	Puerto Datos A4
6	6	Puerto Datos A5
7	7	Puerto Datos A6
8	8	Puerto Datos A7
9	9	Coordenada X, medio o paso completo X_H/~F
10	10	Coordenada Y, medio o paso completo Y_H/~F
11	11	Coordenada X, sentido horario, antihorario X_D/~I
12	12	Coordenada Y, sentido horario, antihorario Y_D/~I
13	13	Carga coordenada X parte baja y media L_X_BM
14	14	Carga coordenada X parte alta L_X_A
15	15	Carga coordenada Y parte baja y media L_Y_BM
16	16	Carga coordenada Y parte alta L_Y_A
17	17	Indica a la PC que ha terminado el control. Fin
18	18	Acarreo en la coordenada X. RCO_X
19	19	Acarreo en la coordenada Y. RCO_Y
20	20	No usado ----
21	21	Inicia control en X. START_X
22	22	Inicia control en Y. START_Y
23	23	Detiene oscilador en X. PARA_MX
24	24	Detiene oscilador en Y. PARA_MY
25	25	Tierra. GND
---	26	Oscilador en X.
---	27	Oscilador en Y.
---	28	Terminó cuenta en X. ACABE_X
---	29	Terminó cuenta en Y. ACABE_Y
---	30	Reloj filtrado para el driver del motor X. RELOJ_X
---	31	Reloj filtrado para el driver del motor Y. RELOJ_Y
---	32	Alimentación. 8 V

---	33	Tierra. GND
---	34	Alimentación. 5 V

Tabla 3.7. Descripción de pines para el DB25 y el Slot de 34 pines.

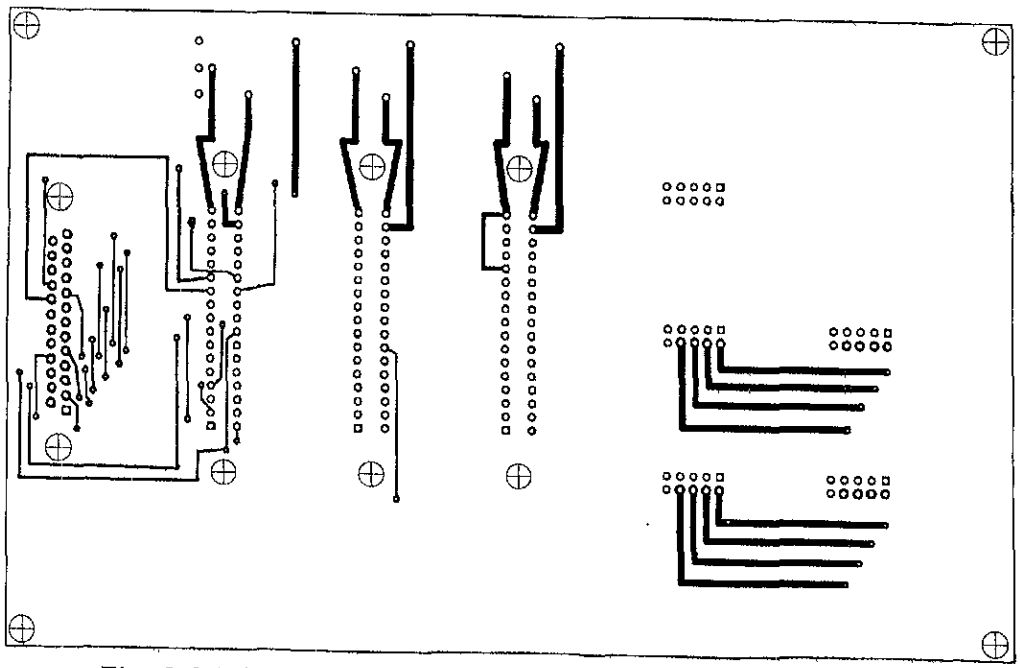


Fig. 3.24. Impreso para la tarjeta madre. Cara de superior.

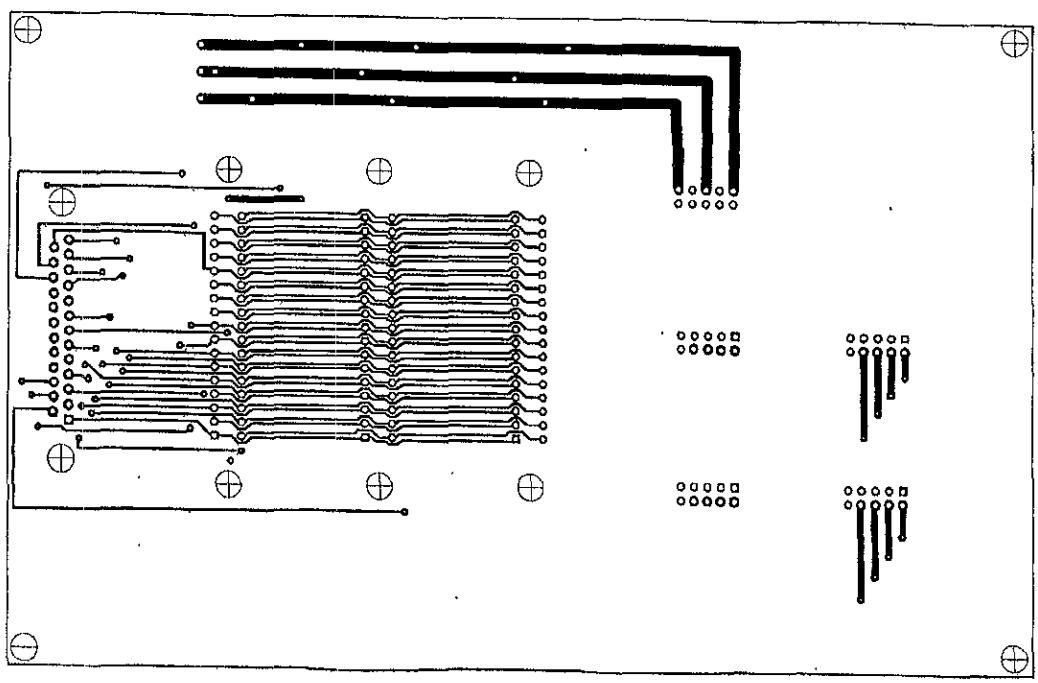


Fig. 3.25. Impreso para la tarjeta madre. Cara de inferior.

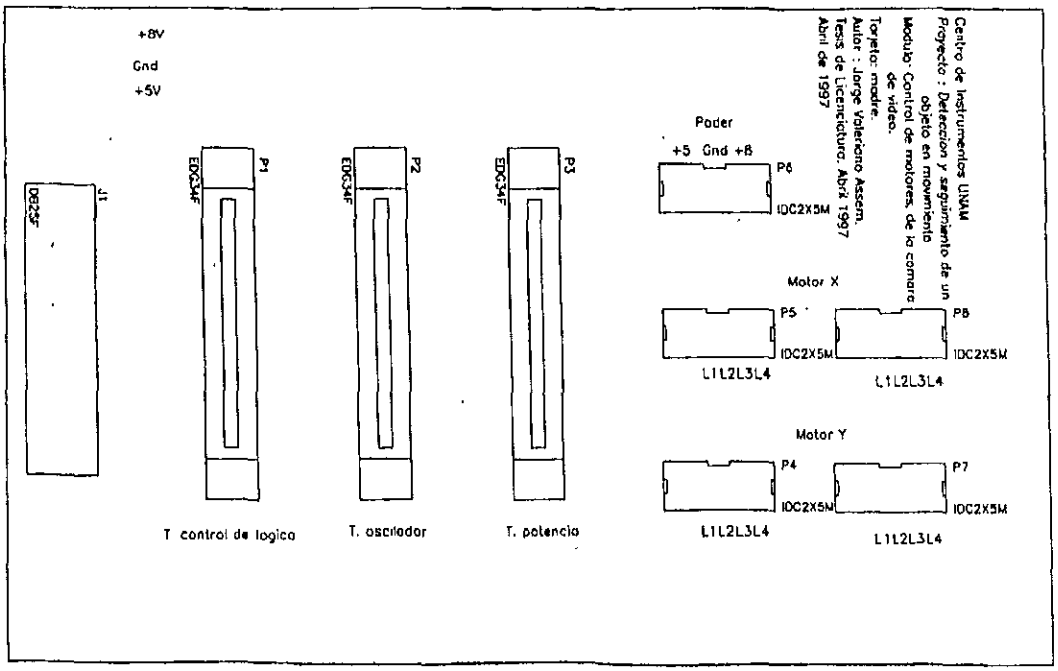


Fig. 3.26. Impreso para la tarjeta madre. Cara de componentes.

3.3.5 Diagrama de tiempo.

La Fig. 3.27 ilustra el diagrama de tiempos para el funcionamiento de la electrónica. Cuando las señales *Enable_X* y *Enable_Y* pasan de nivel 0 a 1, indican que el control de motores ha terminado. Para que la electrónica realice el siguiente movimiento de motores, es necesario enviar señales de control e información referente al número de pasos desde la PC a la electrónica. Las señales *Para_MX* y *Para_MY* deben ser establecidas a 1 para detener el oscilador que asegura el paro de motores, las mismas señales deben regresar a nivel 0 antes de enviar el pulso de inicio de movimiento. En este momento la electrónica está lista para cargar los contadores con los datos calculados por la PC, por lo tanto la información a escribir se envían a través del puerto A, en seguida se habilita a 1 sólo una de las señales de carga de contadores (*L_X_BM, L_X_A, L_Y_BM, L_Y_A*) acorde al orden de carga deseado. Paralelamente a la carga de datos se establecen las señales *X_H/F, Y_H/F, X_D/I* y *Y_D/I*, a sus niveles deseados según el tamaño de paso y sentido de giro. Finalmente, las señales *Start_X* y *Start_Y* generan un pulso alto con el fin de indicar a la electrónica que inicie el movimiento. En este momento los relojes para los Drivers comienzan a oscilar y se detienen cuando las señales *Enable_X* y *Enable_Y* pasan de 0 a 1.

3.3.6 Fuente de alimentación.

Se diseñó una sola fuente de alimentación tanto para la lógica como para la etapa de potencia. Esta fuente provee voltajes fijos de 8V hasta 2 A para la parte de

potencia y, de 5V hasta 1 A a la parte digital. Es básicamente un filtro tipo "C", el diagrama esquemático se ilustra en la Fig. 3.28.

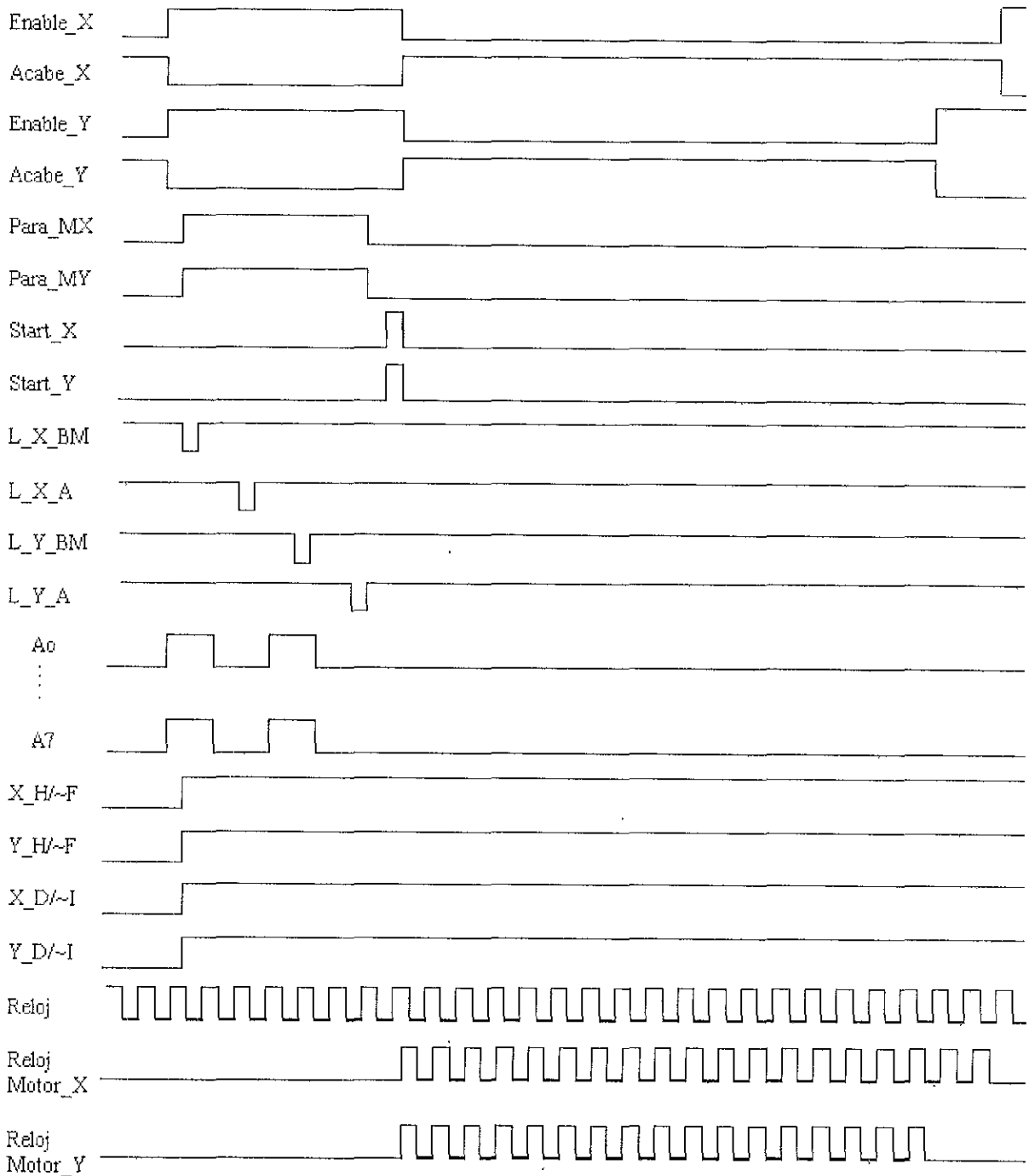


Fig. 3.27. Diagrama de tiempo.

Dispone de un transformador con V_{rms}=18 V y f=60 Hz., en consecuencia V_M=25.45 V.

Requiere las siguientes especificaciones:

- Voltaje de DC de hasta V_{dc}=25.45 - 2(0.7)= 24.05
- Corriente máxima de carga de hasta I_{dc}= 2 A.
- Voltaje de rizo de hasta V_{rpp}= 1 V

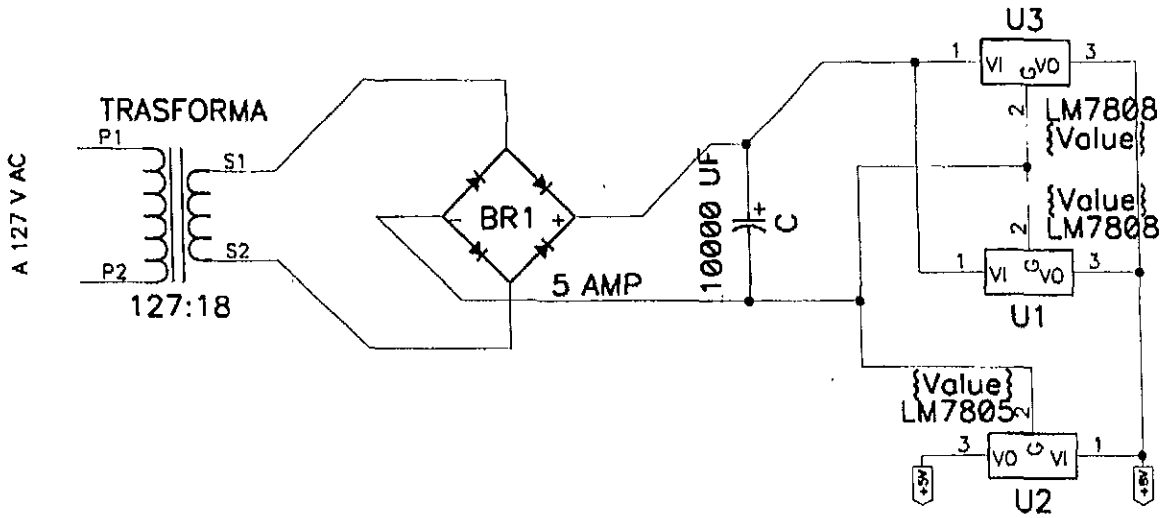


Fig. 3.28. Esquemático para la fuente de alimentación.

El valor del capacitor queda determinado por la siguiente expresión:

$$V_{rpp} = \frac{I_{dc} * V_{dc}}{2fC * V_M}$$

de donde:

$$C = \frac{I_{dc} * V_{dc}}{2f * V_{rpp} * V_M}$$

Sustituyendo se tiene C=15749 uf. Se utilizó un capacitor comercial de 10000 uf logrando un V_{dc}=15.27 V, suficiente para nuestra aplicación. Este V_{dc} de salida es alimentado a dos reguladores LM7808 conectados en paralelo que regulan cada uno a 8 V y 1 A. La salida de 8V se alimenta a un regulador LM7805 que regula a 5 V y 1 A, de esta forma la fuente proporciona 8 V hasta 2 A y 5 V hasta 1 A.

El circuito impreso para la fuente se ilustra en la Figs. 3.29 y 3.30 para las capas inferior y de componentes.

La Fig. 3.31 muestra físicamente la electrónica de control de motores diseñada en circuito impreso y, constituida como se describió en las secciones anteriores.

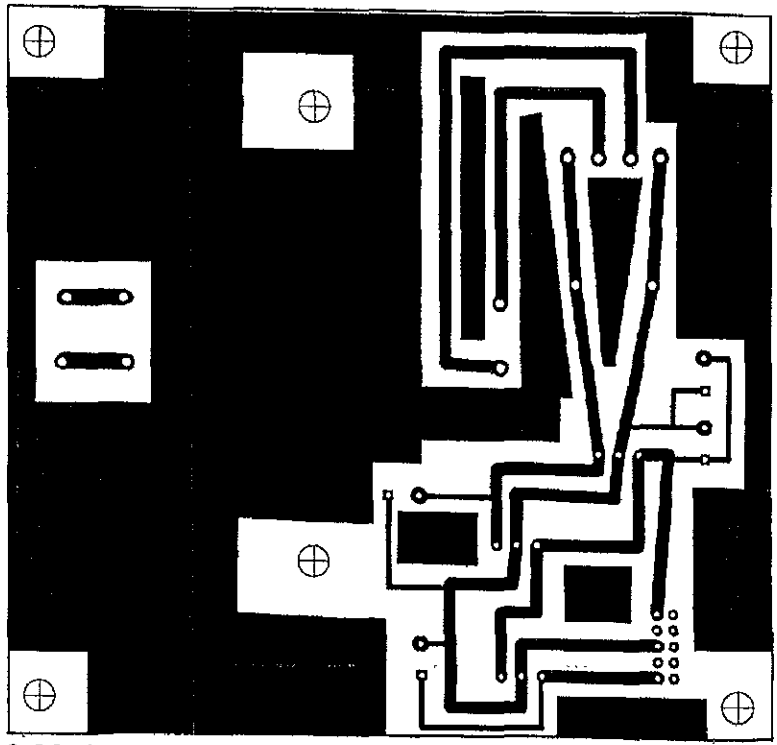


Fig. 3.29. Impreso para la tarjeta de alimentación. Cara inferior.

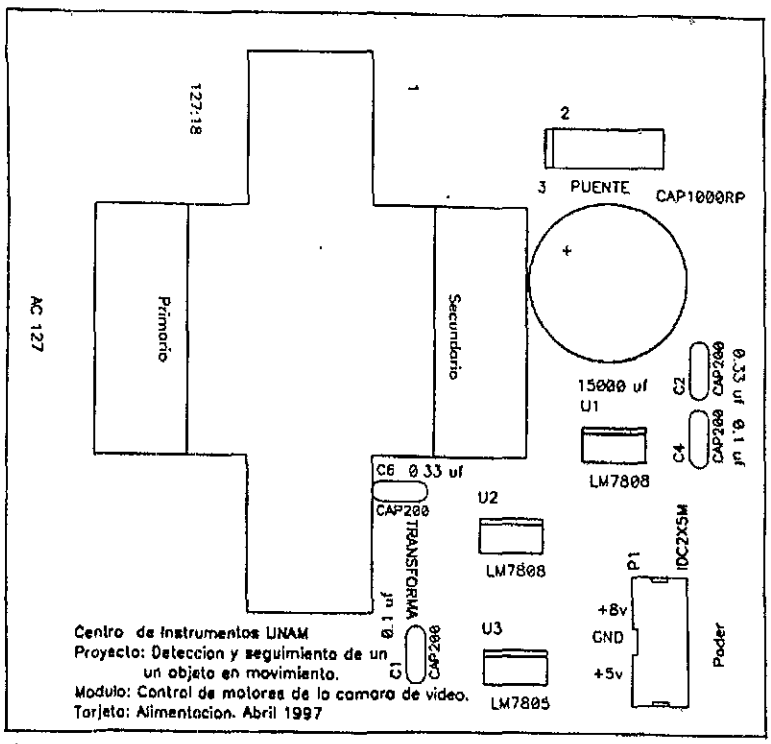
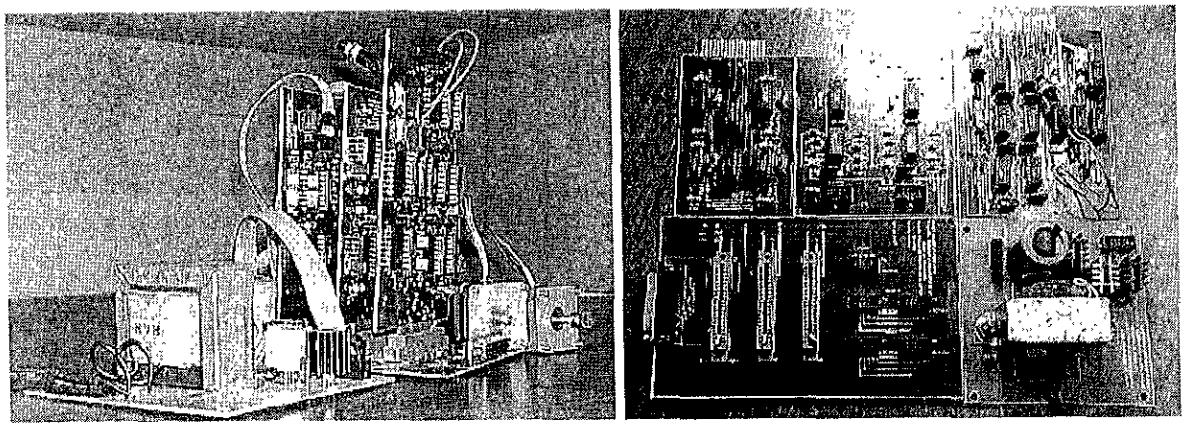


Fig. 3.30. Impreso para la tarjeta de alimentación. Cara de componentes.



(a) (b)

Fig. 3.31. Electrónica de control de motores. (a) Tarjetas ensambladas entre si. (b) Tarjetas por separado.

3.3.7 Software para el funcionamiento del Sistema Físico global.

El Sistema Físico global para el seguimiento de objetos en movimiento, requiere de software que permita la conjunción e interacción de sus tres subsistemas:

- Adquisición y digitalización de imágenes. (Cámara CCD y Frame Grabber Matrox Meteor).

- Detección y seguimiento del objeto en movimiento. (Procesamiento Digital de las imágenes adquiridas y CLB para el seguimiento del objeto).

- Estructura mecánica y electrónica de control para el movimiento de la cámara CCD.

El subsistema dos ha sido desarrollado en el Capítulo 2. Después de evaluación de desempeño tanto en velocidad de procesamiento como en inmunidad a detección de ruido se seleccionó la técnica de detección de bordes por Comparación de Mínima Distancia para el Procesamiento Digital de la Imagen y, se utiliza un CLB para el seguimiento del objeto, cuya versión final para la obtención de las *acciones de control* emplea el *error*, *variación del error*, *velocidad del objeto* y utiliza una ganancia calculada mediante otro CLB que emplea como variables de entrada la *velocidad* y *aceleración* del objeto.

El subsistema uno, descrito en este Capítulo, emplea rutinas de adquisición proporcionadas por el fabricante en una librería para lenguaje "C" , que permiten obtener y enviar imágenes digitalizadas directamente al sistema de memoria principal de la PC.

El subsistema tres, diseñado en este Capítulo permite desplazar físicamente la cámara de video CCD según las acciones de control proporcionadas por el subsistema dos. La electrónica diseñada es operada a través de escrituras y lecturas a los puertos de la interfaz 8255A, tal y como se ilustra en el diagrama de tiempos de la sección 3.3.5.

El diagrama de flujo para el diseño del software que conjunta los tres subsistemas es ilustrado en la Fig. 3.32. Cada número ubicado a la derecha o izquierda es usado para hacer una breve descripción de las acciones hechas por ese bloque, como sigue:

(1) Se encarga de inicializar el sistema de digitalización (Frame Grabber) abriendo canales de comunicación entre éste y la PC. Aloja memoria dinámica para almacenar la imagen digitalizada, imagen de bordes y coordenadas de bordes. Inicializa la interface periférica (8255A) entre la PC y la electrónica de movimiento de motores. Comprueba y efectúa ajustes iniciales en la electrónica para el movimiento de motores. La Fig. 3.33 ilustra el diagrama de flujo desglosado para este bloque.

(2) Utiliza el Frame Grabber para adquirir una imagen digitalizada y la transfiere al sistema de memoria principal. A continuación efectúa una detección de bordes almacenando en memoria principal la imagen resultante, así como las coordenadas de éstos. Finalmente calcula el centroide del objeto relativo al campo de visión de la cámara usando las coordenadas de los bordes. La Fig. 3.34 ilustra el diagrama de flujo desglosado para este bloque.

(3) Utiliza la historia de centroides proporcionada por el bloque anterior para calcular las variables de entrada al CLB. Este genera una acción de control en pixeles que es la posición de la cámara. Por lo tanto esta posición requiere ser transformada a pasos de motor para poder ser aplicada a los motores que accionan la estructura mecánica. La Fig. 3.35 ilustra el diagrama de flujo desglosado para este bloque y la Fig. 3.38 para el CLB utilizado en el cálculo de los parámetros de ganancia.

(4) Este bloque transforma la Acción de Control proporcionada por el CLB, cuyas unidades son pixeles, a un número de pasos que debe girar el motor. Esta transformación requiere, en primer lugar, parámetros como son el equivalente en grados de cada paso del motor (TP), la relación de la caja de engranes (FR) y, en segundo lugar, una calibración que permita conocer la relación de conversión de

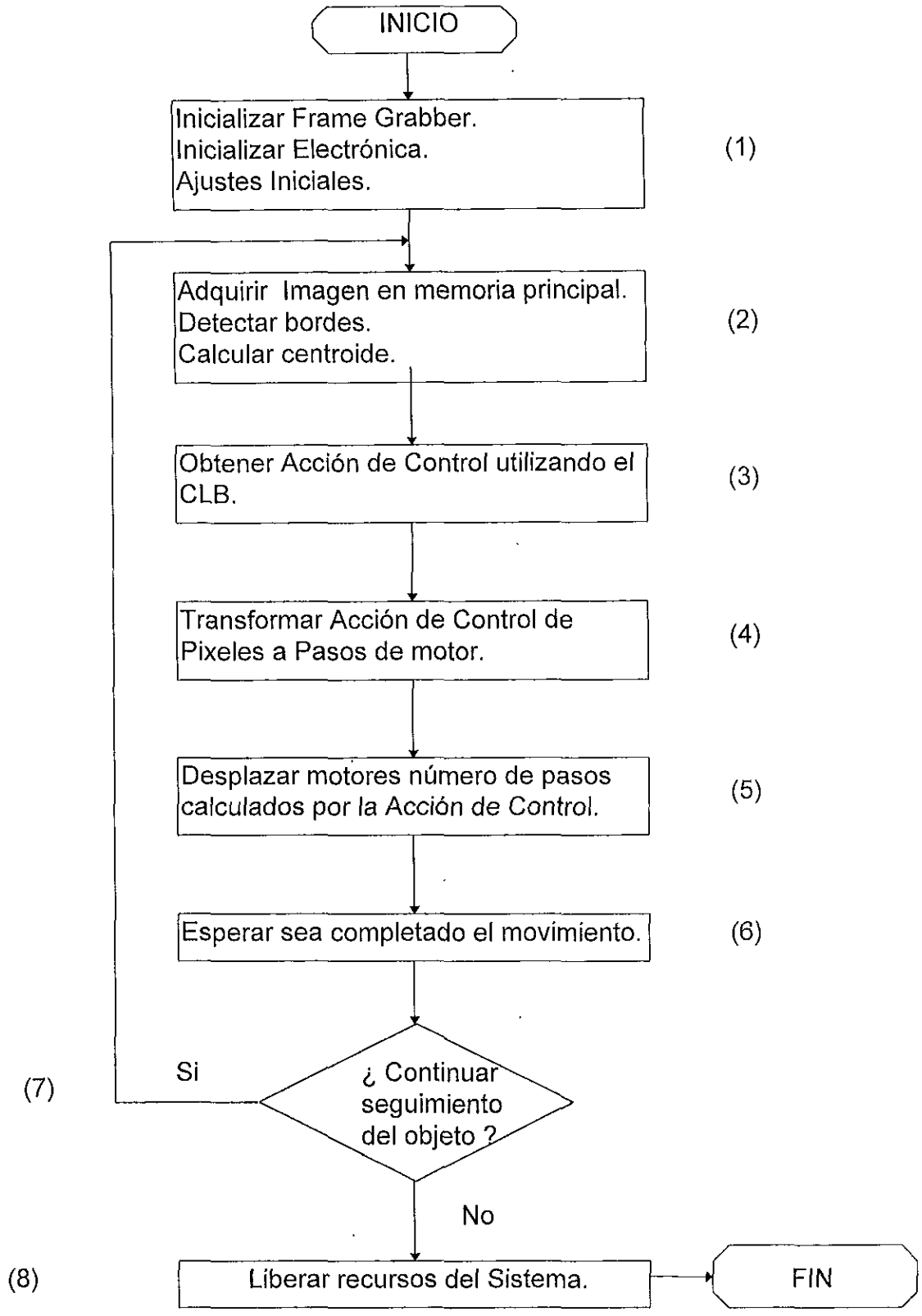


Fig. 3.32 Diagrama de flujo que conjunta el Sistema Físico Global.

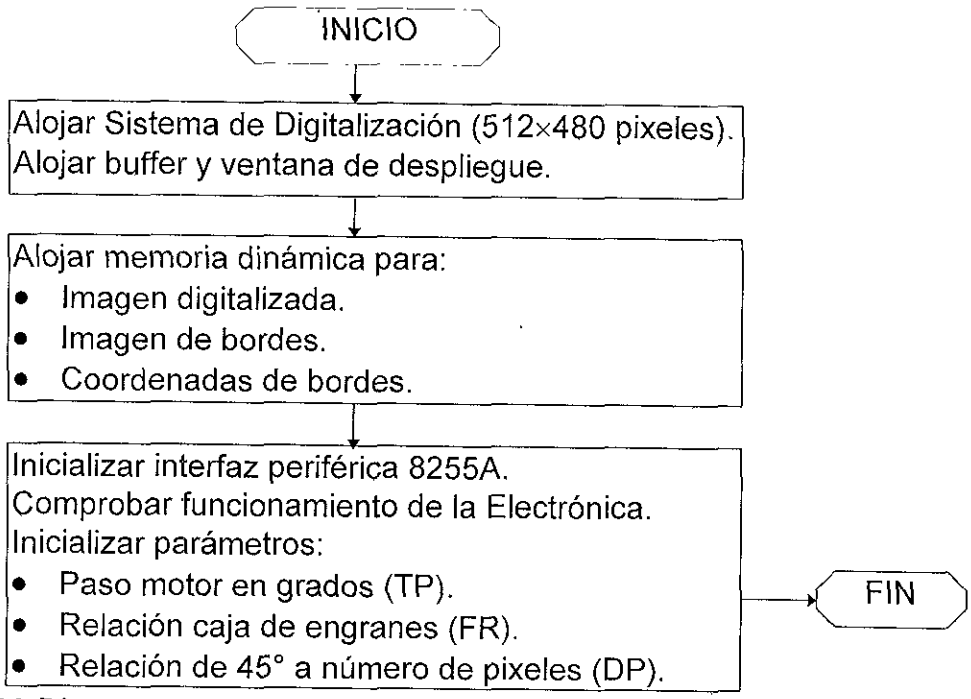


Fig. 3.33 Diagrama de flujo desglosado para la Fig. 3.32 bloque (1).

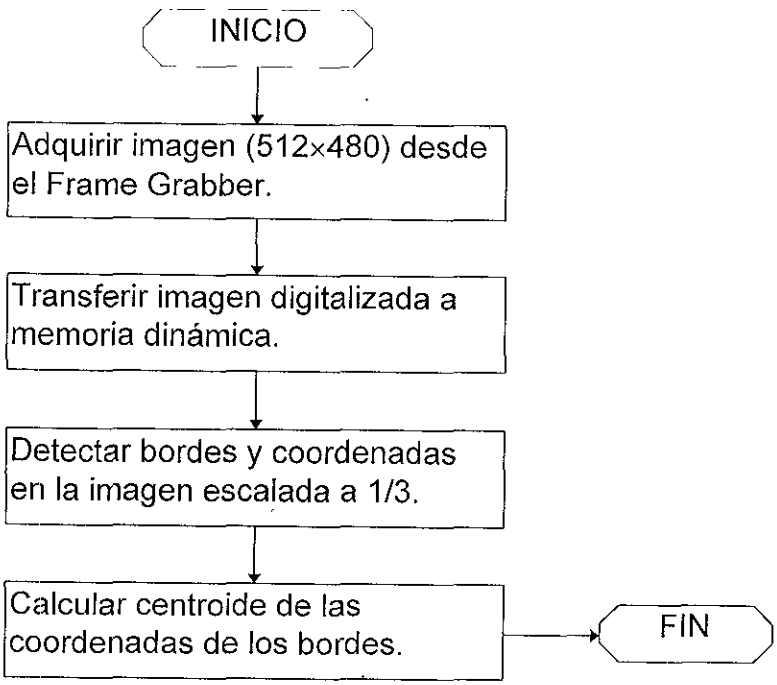


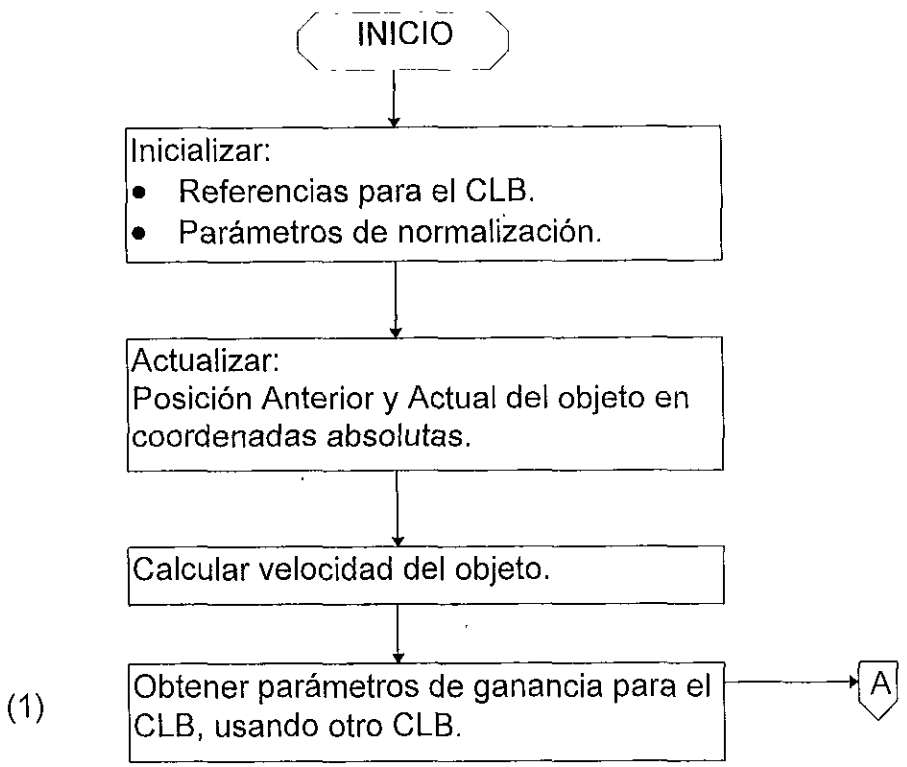
Fig. 3.34 Diagrama de flujo desglosado para la Fig. 3.32 bloque (2).

pixeles a pasos de motor. Esto se lleva a cabo en el bloque (1) de la Fig. 3.32 y en forma desglosada en la Fig. 3.33.

La calibración para la conversión de pixeles a pasos de motor se efectúa de la siguiente manera: se adquiere una imagen A y se calcula su centroide CA, posteriormente se desplaza la cámara 45° y se adquiere una imagen B calculando su centroide CB. La diferencia en pixeles DP= | CB-CA | es equivalente a 45°. De manera que este bloque utiliza esta relación para transformar la Acción de Control dada en pixeles (ACP) a grados, la cual es escalada por el factor de relación de la caja de engranes (FR) y, finalmente se transforma a número de pasos usando el numero de grados por paso (TP). Así la transformación de ACP a pasos (ACPP) se obtiene como:

$$ACPP=(((45\times ACP)/DP)\times FR)/TP$$

(5) Utiliza la Acción de Control en pasos de motor (ACPP), para determinar los parámetros de sentido y tamaño de paso (paso completo o medio paso). Transfiere a la electrónica la información de número de pasos, señales de control e inicia el movimiento. La Fig. 3.36 ilustra el diagrama de flujo desglosado para este bloque.



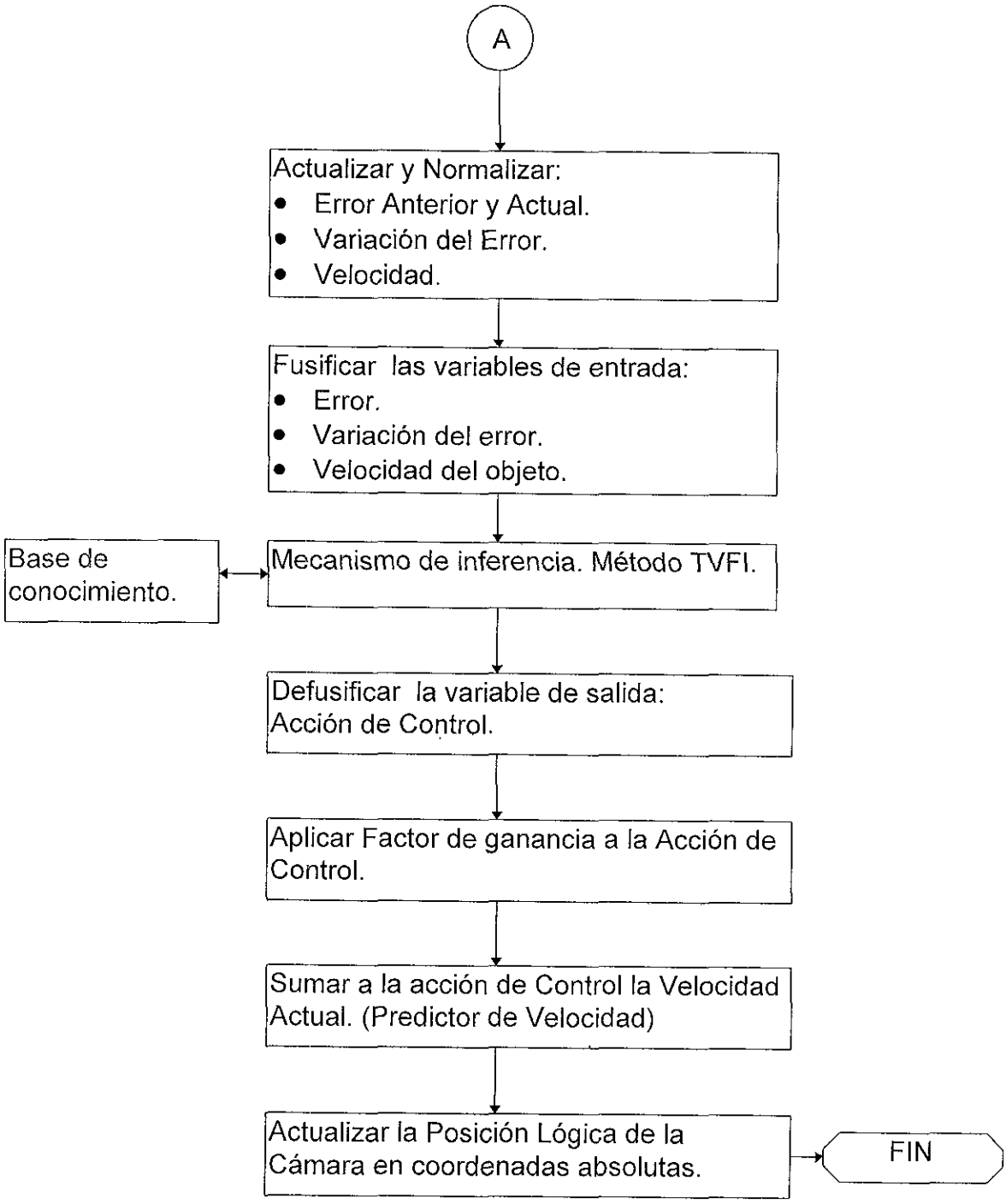


Fig. 3.35 Diagrama de flujo desglosado para la Fig. 3.32 bloque (3).

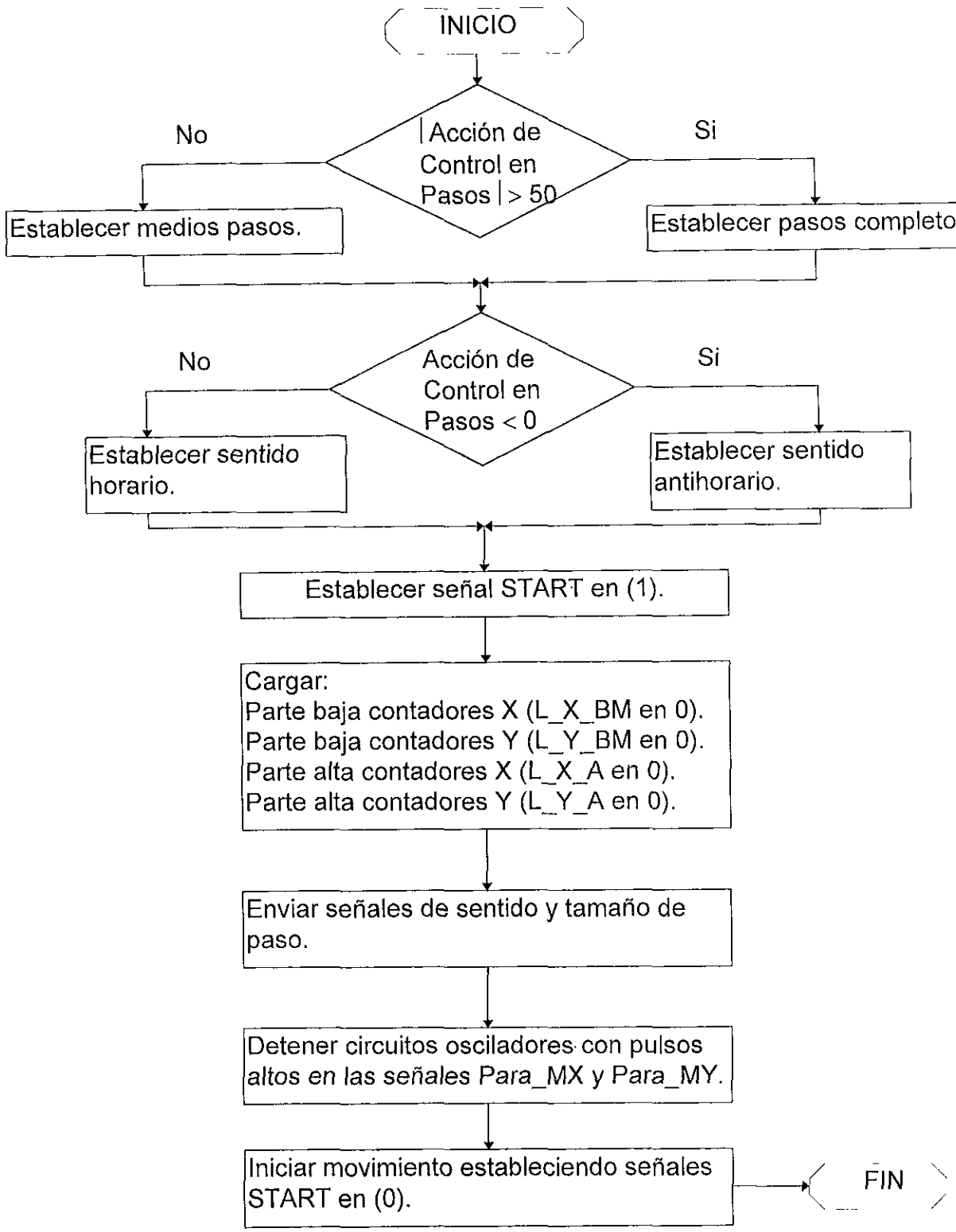


Fig. 3.36 Diagrama de flujo desglosado para la Fig. 3.32 bloque (5).

(6) Este bloque verifica continuamente el estado de la electrónica y espera hasta que se halla completado el movimiento. Esto se logra verificando periódicamente la señal FIN en el bit cero del puerto C del 8255A.

(7) Se verifica si se desea continuar con el seguimiento del objeto. Este proceso se puede interrumpir pulsando cualquier tecla.

(8) Libera los recursos utilizados por el Frame Grabber cerrando los canales de comunicación entre éste y la PC. Libera la memoria dinámica utilizada para la imagen adquirida, imagen de bordes y coordenadas de bordes. La Fig. 3.37 ilustra el diagrama de flujo desglosado para este bloque.

El bloque (1) de la Fig. 3.35, calcula los factores de ganancia para el CLB utilizando otro CLB que emplea como variables de entrada la *velocidad* y *aceleración* del objeto. En la Fig. 3.38 se desglosa este CLB adicional.

Se hace notar que los algoritmos descritos en los diagramas de flujo anteriores, por ejemplo para el CLB, no hacen referencia especifica a la coordenada X o Y que se esté controlando, porque el procedimiento es idéntico para cada una de ellas. Se asume que tales algoritmos son aplicados por coordenada. Sólo dónde es necesario especificar la coordenada de que se trata se hace referencia a ellas, como es el caso de la Fig. 3.36 donde se define la secuencia para cargar la información en los contadores.

El código en Borland C 4.5 para el algoritmo descrito en los diagramas de flujo anteriores así como el diccionario de datos se muestra en el Apéndice B de esta tesis.

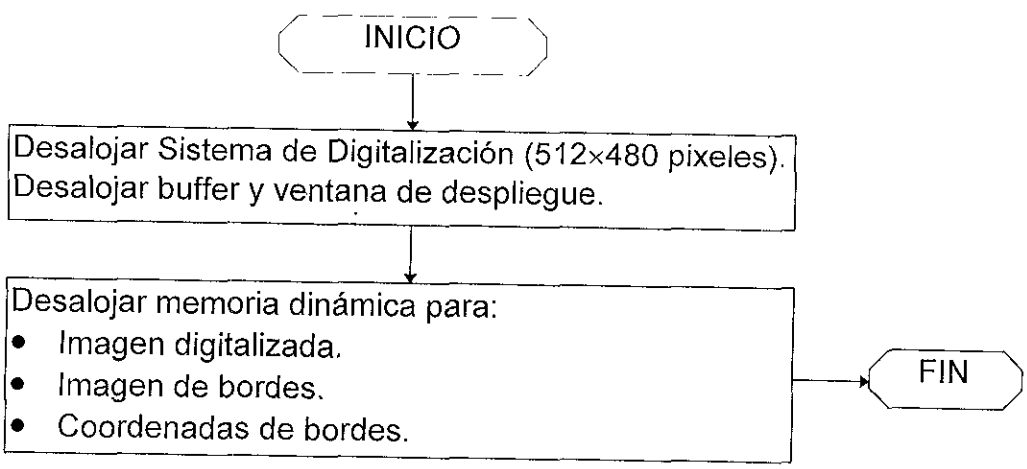


Fig. 3.37 Diagrama de flujo desglosado para la Fig. 3.32 bloque (8).

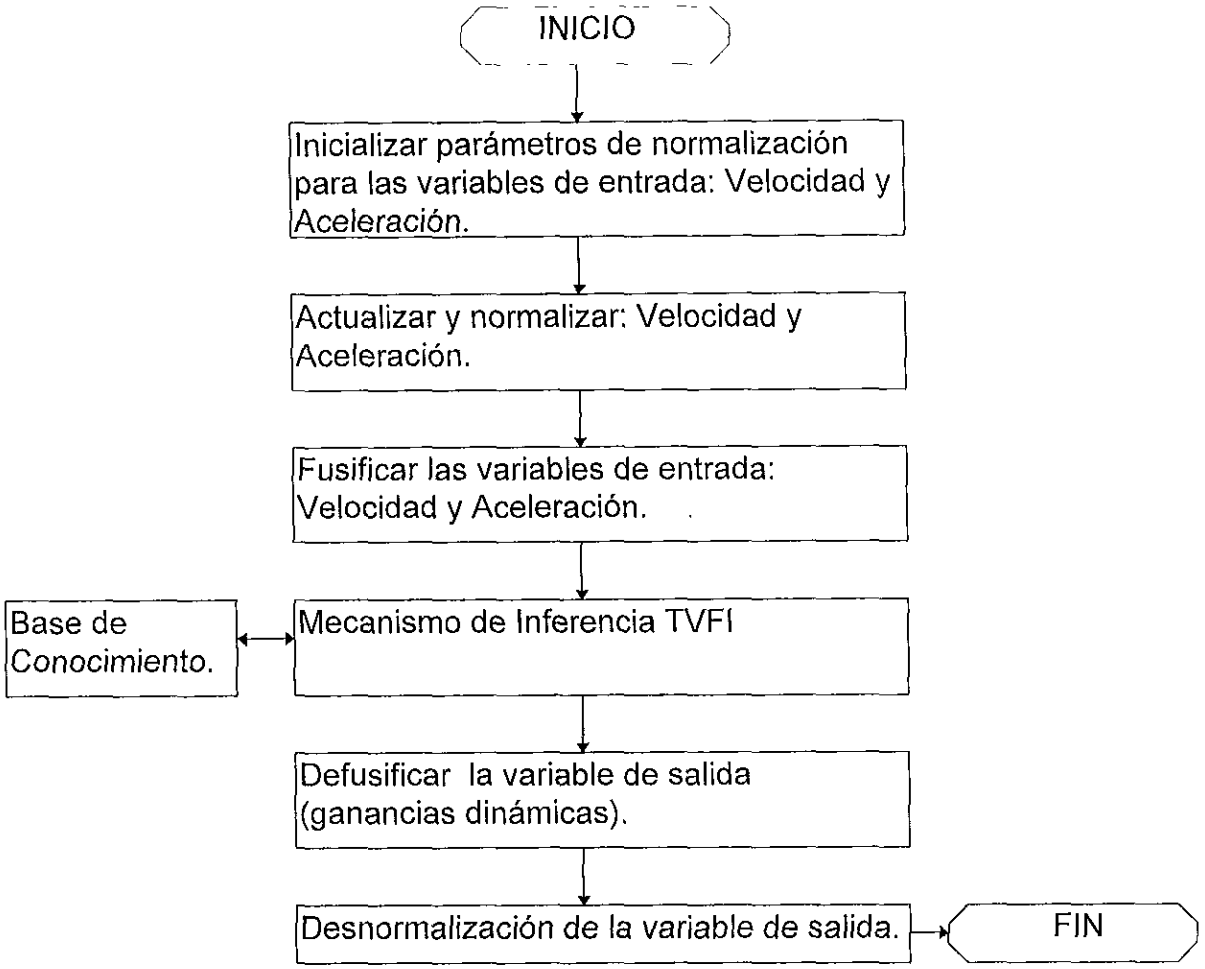


Fig. 3.38 Diagrama de flujo desglosado para la Fig. 3.35 bloque (1).

CONCLUSIONES Y RECOMENDACIONES.

En este trabajo, para el seguimiento de un objeto en movimiento, requerimos diseñar un controlador que usando la historia de los centroides del objeto, permita generar la acción de control para la posición de la cámara. Nuestra planta está constituida por la estructura mecánica, cámara CCD, electrónica de control de motores y objeto en movimiento, el cual puede desplazarse aleatoriamente tanto en trayectoria como en velocidad. Por tanto, construir un modelo matemático del comportamiento de la planta es prácticamente imposible. En este punto, la lógica borrosa y especialmente su aplicación en control, representa una herramienta útil en el diseño de controladores para plantas de las cuales no conocemos su modelo matemático, sino sólo su comportamiento basado en la experiencia.

El control borroso permitió resolver este problema conjuntando, dentro de una base de reglas que forma parte de las etapas de razonamiento de un CLB, las observaciones de los posibles comportamientos del objeto en movimiento, así como las posibles acciones de control.

Dos de los aspectos fundamentales de quienes depende la funcionalidad del CLB son: la generación de la base de reglas y, la sintonización del controlador.

La base de reglas permite reunir el conocimiento que contendrá el CLB acerca del comportamiento de la planta, a partir del cual es posible generar un conjunto de resultados mediante los cuales se produce la acción de control. La construcción de las reglas depende de la experiencia que se tenga del comportamiento del sistema. Inicialmente se propuso un CLB-PI que usa el error y

su derivada para generar una acción de control, la que pretende aplicar acciones drásticas tanto para valores grandes de error como de su variación. A pesar de los resultados satisfactorios de la Fig. 2.23 en comparación con una Corrección Simple del Error en la Fig. 2.8, tenemos problemas de oscilación fuera del valor de referencia. La referencia se ajusta fácilmente incorporando el predictor de velocidad (Fig. 2.24); sin embargo, el controlador es incapaz de disminuir la oscilación a pesar de la sintonización de parámetros.

No obstante, el control borroso ofrece la alternativa de incorporar, relativamente fácil, múltiples modificaciones para mejorar su desempeño, como puede ser la incorporación de nuevas variables de entrada. En este momento el controlador deja de ser un control típico que utiliza el error y su derivada.

En nuestro caso, como el objeto no sigue un patrón de velocidad constante en su desplazamiento, se determinó la incorporación de la variable *velocidad* al conjunto de entradas del CLB, con el objetivo de minimizar la acción de control a velocidades bajas y, en el caso de sentido contrario entre el desplazamiento del objeto y la acción de control; maximizando esta acción en los casos contrarios. Esta mejora genera las respuestas de la Fig. 2.31 que superan notablemente las de la Fig. 2.24.

La sintonización es una etapa clave y compleja en el diseño de un CLB, porque son muchas características susceptibles de modificarse para lograrlo. La sintonización depende de los parámetros de normalización, reglas y funciones de membresía. Algunos con efecto global como los parámetros de normalización y otros con efecto puntual como las reglas. En el CLB-PI propuesto inicialmente, fue necesario sintonizar los parámetros de normalización, reglas y simetría de las funciones de membresía con base en la observación del comportamiento del sistema. Sin embargo, en el CLB que incorpora la variable de entrada *velocidad,* fue necesario sintonizar, adicionalmente, el tipo de función de membresía a una forma exponencial.

La sintonización de los parámetros de normalización es tan importante, que la capacidad del CLB para ajustar automáticamente uno o más de ellos en función de las circunstancias que en ese momento imperen, pueden incrementar notablemente su desempeño. En nuestro caso, se hizo un ajuste del parámetro de ganancia utilizando otro CLB que utiliza como variables de entrada la *velocidad* y *aceleración* del objeto, con el fin de minimizar la ganancia a velocidades bajas y aceleraciones decrecientes y, maximizarla en los casos contrarios. Esta modificación proporciona al CLB la característica de adaptabilidad. Los resultados ilustrados en la Fig. 2.38 superan a las respuestas de la Fig. 2.31.

Por parte de la sección para el Procesamiento Digital de la Imagen, dos aspectos fundamentales fueron seleccionados para decidir el detector de bordes a utilizar:

- Velocidad de procesamiento.
- Inmunidad al ruido.

La técnica de Comparación de Mínima Distancia presentó el mejor equilibrio entre ambos requisitos, porque su sencilla programación requiere pocos recursos de computo y, su inmunidad al ruido en función del parámetro *radio* permite un cálculo preciso del centroide. La técnica Borrosa es buen detector de bordes e inmune al ruido pero, consume muchos recursos de computo. La técnica de Laplace es un excelente detector de intensidad de bordes y en consecuencia sensible al ruido.

En relación a la electrónica para el control de motores, los circuitos diseñados son prototipos cuya función es ejecutar la acción de control liberando completamente el procesador de la PC de esa tarea. Un aspecto delicado en nuestra electrónica fue la susceptibilidad a fuentes electrostáticas, debido al uso de un flip-flop disparado por flanco para controlar el estado de los motores. El problema se resolvió diseñando un circuito oscilador que comienza a oscilar el pin de limpieza del flip-flop cuando la cuenta ha terminado, evitando se modifique el estado de los motores.

La electrónica fue diseñada modularmente de manera que es posible cambiar los diseños de la etapa de potencia, del oscilador y lógica para el conteo de pasos, sin afectar las demás etapas. El bus de la tarjeta madre posee pines no utilizados para un posible uso futuro.

Se recomiendan los siguientes puntos para trabajos posteriores:

- Explorar herramientas como Redes Neuronales para el sistema de control de seguimiento del objeto.

- Optimizar el algoritmo Borroso y de Comparación de Mínima Distancia en el sistema de Procesamiento Digital de la Imagen.

- Desarrollar un digitalizador de propósito específico para el sistema de adquisición de imágenes que permita a la aplicación, independencia de la plataforma Windows, para poder crear un sistema autónomo.

- Diseño en hardware de los mejores algoritmos de control de seguimiento del objeto y procesamiento digital de la imagen, tendiente a procesar 30 cuadros por segundo (estándar NTSC).

- Incorporación de la electrónica de control de motores en un microcontrolador.

- Conjunción de un sistema completamente autónomo.

REFERENCIAS BIBLIOGRÁFICAS.

[1] Ballard H., Dana and Brown M., Christopher. Computer Vision. Ed. Pretince-Hall. USA, 1982.

[2] Fu K., S. et al. Robotics: Control, Sensing, Vision and Intelligence. Ed. McGraw-Hill. USA, 1987.

[3] Vermeyen., P. et al. Visual Feedback for a Robot to Recognize and Pick up Parts. The First IFAC Workshop Espoo. Finland, 10-12 june 1986. P. 109. Edited by M. Ollus.

[4] Andersen D., J. Robot Control Based on Neighborhood Image Operations. The First IFAC Workshop Espoo. Finland, 10-12 june 1986. P. 99. Edited by M. Ollus.

[5] Nielsen., L. A Visual Servo. The First IFAC Workshop Espoo. Finland, 10-12 june 1986. P. 89. Edited by M. Ollus.

[6] Castillo H., J. Diseño basado en Electrónica Analógica de las Etapas que conforman un Controlador Borroso. Universidad Nacional Autónoma de México, División de estudios de posgrado. 1997.

[7] Zadeh A., L. Fuzzy Sets. Information and Control, Vol. 8, P.p. 338-353, 1965.

[8] Von Altrock., C. et al. Advanced Fussy Logic Control Technologies in Automotive Aplication. IEEE Internacional Conference on Fuzzy Systems, P.p. 835-842, 1992.

[9] Takagi., H. Survey of Fuzzy Logic Application in Image-Processing Equipment. Indistrial Application of Fuzzy Logic and Intelligent Systems, chapter 4, IEEE Press, 1995.

[10] Zadeh A., L. Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. IEEE Trans. Syst., Man., Cybern., Vol. SMC-3, No. 1, P.p 28-44 Jan 1973.

[11] Jamshidi Vadiee., Ross. Fuzzy Logic and Control Software and Hardware Applications. 1st edition, Ed. Pretince Hall. USA, 1993.
[12] Yager., R. y Filev., D. Essentials of Fuzzy Modeling and Control. Ed. John Wiley & Sons Inc. USA, 1994.
[13] Gulley., N. and Jang., R. Fuzzy Logic Toolbox for use with MATLAB. User's guide. The Math Works Inc. USA, 1995.
[14] Zheng., Li. A practical Guide to Tune of Proportional and Integral (PI) like Fuzzy Controllers. IEEE International Conference on Fuzzy Systems. P.p. 633, 1992.
[15] González C., Rafael and Wintz., Paul. Digital Image Processing. Ed. Addison-Wesley. Second Edition, 1987.
[16] Russo., Fabrizio. A user-Friendly Research Tool for Image Processing with Fuzzy Rules. IEEE International Conference on Fuzzy Systems, P.p. 561, 1992.
[17] Matrox Electronic Systems Ltd. User Guide, Command Reference and Board Specific Notes. Manual No. 10514-MU-0200, December 20, 1996. Printed in Canada.
[18] Matrox Electronic Systems Ltd. Matrox Intellicam User Guide Version 2.0. Manual No. 10521-MU-0300, December 20, 1996. Printed in Canada.
[19] Matrox Electronic Systems Ltd. Matrox Meteor Installation and Hardware Reference. Manual No. 10529-MT-0100, February 20, 1997. Printed in Canada.
[20] Accel Technologies Inc. Tango PRO Schematic Refernce. USA 1993.
[21] Accel Technologies Inc. Tango PRO Using Schematic. USA 1993.
[22] Accel Technologies Inc. Tango PRO PCB Reference. USA 1993.
[23] Accel Technologies Inc. Tango PRO Using PCB. USA 1993.
[24] Accel Technologies Inc. Tango PRO Using the Library Manager. USA 1993.

APÉNDICE A
DICCIONARIO DE DATOS Y CÓDIGO FUENTE DEL SIMULADOR PARA EL SEGUIMIENTO DE UN OBJETO EN MOVIMIENTO.

DICCIONARIO DE DATOS.

Tabla A.1

Variable o Función	Tipo	Descripción
posx, posy	Entero. Privado.	Posición donde se visualiza la cámara.
imagen	Apuntador a caracter no signado. Privado.	Apuntador al primer elemento de la imagen capturada por la cámara.
pos_act_ren	Entero.	Posición actual de la cámara en renglones.
pos_act_col	Entero.	Posición actual de la cámara en columnas.
num_ren	Cartacter no signado.	Número de renglones de la cámara (alto).
num_col	Cartacter no signado.	Número de columnas de la cámara (ancho).
camara(int nr, int nc)	----	Constructor.
~camara(void)	----	Destructor.

prepara(unsigned char nr, unsigned char nc, int actual_x, int actual_y);	Void.	Prepara tamaño y posición de la camara.
captura(void);	Apuntador a caracter no signado.	Captura imagen.

Tabla A.1. Clase camara.

Tabla A.1.1

Variable	**Tipo**	**Descripción**
icono_camara	Arreglo caracter.	Ubicación del archivo con el icono de la cámara.
icono_salir	Arreglo caracter.	Ubicación del archivo con el icono "salir".
icono_trayectoria	Arreglo caracter.	Ubicación del archivo con el icono de la "trayectoria".
icono_t_objeto	Arreglo caracter.	Ubicación del archivo con el icono para periodo de movimiento del objeto.
icono_t_camara	Arreglo caracter.	Ubicación del archivo con el icono para periodo de captura de la cámara.
icono_figura1	Arreglo caracter.	Ubicación del archivo con el icono de la figura 1.
icono_figura2	Arreglo caracter.	Ubicación del archivo con el icono de la figura 2.
icono_figura3	Arreglo caracter.	Ubicación del archivo con el icono de la figura 3.
icono_figura4	Arreglo caracter	Ubicación del archivo con el icono de la figura 4.

Tabla A.1.1. camara.prepara

Tabla A.2

Variable o Función	**Tipo**	**Descripción**
num_ren	Caracter no signado. Privado.	Número de renglones de la imagen.
num_col	Caracter no signado. Privado.	Número de columnas de la imagen.
imagen	Apuntador a caracter no signado.	Apuntador al primer elemento la imagen.
pos_act_ren	Entero.	Posición actual de la imagen en renglones.

pos_act_col	Entero.	Posición actual de la imagen en columnas.
x,y	Apuntadores a caracter no signado.	Coordenadas de la trayectoria en columna, renglón.
objeto(void)	---	Constructor.
~objeto(void)	---	Destructor.
lee(char * camino, unsigned char nr, unsigned char nc)	Void.	Lee imagen de tamaño "nr×nc" de un archivo ubicado en "camino".
mover(int renglon, int columna,char bandera)	Void.	Mueve imagen de posición actual a nuevo "renglon", "columna".
poner(int renglon, int columna)	Void.	Coloca imagen en posición indicada por "renglon" "columna".
elemento(unsigned char huge * arreglo,unsigned int elemento, unsigned int valor, char acceso)	Entero no signado.	Lee o escribe en el "elemento" de "arreglo" el dato "valor", según "acceso".
trayectoria(void)	Void	Traza trayectoria.
linea(int x1, int y1, int x2, int y2, int *indice,unsigned char huge *x,unsigned char huge *y,int color)	Void	Traza una recta entre los puntos (x1,y1) y (x2,y2) y almacena los puntos en los arreglos de trayectoria "x" e "y" a partir de "índice".

Tabla A.2. Clase objeto.

Tabla A.2.1

Variable	Tipo	Descripción
tempo	Apuntador a archivo.	Apunta al archivo que contiene el objeto a seguir.
auxiliar	Apuntador a caracter no signado.	Apuntador para desplazarse sobre la imagen elida.

Tabla A.2.1. objeto.lee

Tabla A.2.2

Variable	Tipo	Descripción
prueba, apunta	Apuntadores a caracter no signados.	Apuntadores utilizados para eliminar las partes de la imagen no sobre escrita.

Tabla A.2.2. objeto.mover

Tabla A.2.3

Variable	Tipo	Descripción
bit_ini, bit_fin	Entero largo no signado.	Bits entre los que se halla el dato.
byte_ini, byte_fin	Entero no signado.	Bytes que contienen la información.
bit_fin_byte_fin	Entero no signado.	Bits dentro de los que se halla el dato en los bytes que contienen la información.
temporal, mascara, desplaza	Enteros no signados.	Variables para leer o escribir la información.
datos	Entero no signado.	Contiene los bytes de interés en formato entero.
valor_aux	Entero no signado.	Variable auxiliar.

Tabla A.2.3. objeto.elemento

Tabla A.2.4

Variable	Tipo	Descripción
xs,ys,xa,ya	Enteros.	Posiciones actual y anterior del pixel señalado.
indice	Entero.	Indice al elemento siguiente a accesar.
contador	Entero.	Variable contador.
num_c	Entero.	Número de puntos a graficar para trayectoria constante.
num_l	Entero.	Número de puntos a graficar para trayectoria lineal.
a,b	Enteros.	Constantes para los incrementos en "x" e "y".
trayectoria	Caracter.	Tipo de trayectoria.

Tabla A.2.4. objeto.trayectoria

Tabla A.2.5

Variable	Tipo	Descripción
xi,yi	Enteros.	Punto inicial para trazado de la línea.
i,j,k	Enteros.	Contadores.
m	Flotante.	Pendiente de la línea.
bandera	Caracter.	Indica como se calcula la recta y=f(x) o x=f(y)

Tabla A.2.5. objeto.linea.

Tabla A.3

Variable o Función	Tipo	Descripción
e1x,e2x	Flotante. Privado.	Error actual(2) y anterior(1) en "x" (columnas) para el CLB.
e1y,e2y	Flotante. Privado.	Error actual(2) y anterior(1) en "y" (renglones) para el CLB.
dex,dey	Flotante. Privado.	Derivadas (variaciones) del error en "x" e "y" para el CLB.
vel1x,vel1y	Flotante. Privado.	Variable "velocidad" en "x" e "y" para el CLB.
xs	Flotante. Privado.	Parámetro para la variación de la simetría de las funciones de membresía para las variables de entrada: "error" y "variación error" al CLB, en el intervalo (0,1).
xv	Flotante. Privado.	Parámetro para la variación de la forma de las funciones de membresía para la variable de entrada "velocidad" al CLB en el intervalo (-1,1).
alfa, beta, corrimiento	Flotante. Privado.	Parámetros para las funciones de membresía exponenciales de la variable de entrada "velocidad" al CLB .
velocidadx, aceleracionx, velocidady, aceleraciony	Flotante. Privado.	Variables de entrada para el CLB que calcula las ganancias a utilizar por el CLB principal.
metodo(void)	---	Constructor.
tecnica(char *camino,int imagen_nr,int imagen_nc,int camara_nr, int camara_nc, char tec)	Void.	Aplica la técnica especificada en "tec" a la imagen ubicada en "camino" con dimensiones "imagen_nr", "imagen_nc",

		usando una camara de dimensiones "camara_nr", "camara_nc".
bajo(unsigned char x)	Float.	Función de membresía tonos bajos para la técnica de procesamiento de la imagen: pseudo-borrosa.
medio(unsigned char x)	Float.	Función de membresía tonos medios para la técnica de procesamiento de la imagen: pseudo-borrosa.
alto(unsigned char x)	Float.	Función de membresía tonos altos para la técnica de procesamiento de la imagen: pseudo-borrosa.
OR_difusa(float x, float y, float z)	Caracter no signado.	Operador borroso OR para la tecnica pseudo-borrosa de procesamiento de la imagen.
predictor_difuso(int xo, int yo,int v1x, int v1y, int *dux, int *duy)	Void.	Controlador Lógico Borroso (CLB). Recibe las coordenadas del centroide actual, la velocidad del objeto y las usa para calcular el "error" y su "variación". Devuelve las acciones de control en los apuntadores "dux" y "duy".
negativo(float x)	Float.	Función de membresía para las variables de entrada: "error" y "variación error".
zero(float x)	Float.	Función de membresía para las variables de entrada: "error" y "variación error".
positivo(float x)	Float.	Función de membresía para las variables de entrada: "error" y "variación error".
min_and(float a, float b, float c)	Float.	Operador AND borroso para el CLB (selecciona mínimo).
max_or(float a, float b, float c)	Float.	Operador OR borroso para el CLB (selecciona máximo).
positivo_vel(float x)	Float.	Función de membresía lineal para la variable de entrada: "velocidad" al CLB.
negativo_vel(float x)	Float.	Función de membresía lineal para

		la variable de entrada: "velocidad" al CLB.
positivo_vel_e(float x)	Float.	Función de membresía exponencial para la variable de entrada: "velocidad" al CLB.
negativo_vel_e(float x)	Float.	Función de membresía exponencial para la variable de entrada: "velocidad" al CLB.
ganancia_dinamica_FLC(int v1x, int v1y,int *gananciax, int *gananciay);	Void.	Este CLB recibe la velocidad del objeto en ambas coordenadas y devuelve en los apuntadores: "gananciax" y "gananciay" las ganancias debe aplicar el CLB principal.

Tabla A.3. Clase metodo.

Tabla A.3.1

Variable	Tipo	Descripción
filtro	Arreglo bidimencional flotante.	Filtro de Laplace.
M	Entero.	Dimensión del filtro.
coor	Apuntador a caracter no signado.	Coordenadas de los bordes [x,y].
nbordes	Entero no signado.	Número de puntos borde.
A,B	Entero.	Dimensiones de la imagen de entrada. Renglones y columnas.
i,j,k,l	Entero.	Contadores.
suma	Flotante.	Resultado de la convolución.
suma_a	Flotante.	Resultado de la convolución anterior.
y,x	Enteros.	Coordenadas temporales.
pixelf	Flotante.	Valor actual del pixel del filtro.
pixeli	Caracter no signado.	Valor actual del pixel de la imagen.
centroide_int	Enteros no signados.	Centroide entero de la

		imagen.
centroide_float	Flotantes.	Centroide flotante de la imagen.
m	Entero no signado.	Contador.
entrada	Apuntador a caracter no signado.	Apuntador a la imagen a procesar.
auxiliar	Apuntador a caracter no signado.	Apuntador a la imagen capturada.
base	Apuntador a caracter no signado.	Respalda el inicio de auxiliar
object	Objeto.	Objeto.
camare	Camara.	Camara de video.
comienzo	Inicializando.	Objeto para inicializar el modo de video.
t_objeto	Entero.	Periodo del movimiento del objeto en centésimas de segundo.
t_camara	Entero.	Periodo de despliegue de la cámara en centésimas de segundo.
tp	Struct time.	Respaldo del tiempo del sistema.
tl_objeto	Entero largo.	Tiempo limite para efectuar nuevo movimiento del objeto en centésimas de segundo.
tl_camara	Entero largo.	Tiempo límite para efectuar nuevo despliegue de la camara en centésimas de segundo.
indice	Entero largo.	Indice del arreglo trayectoria.
opcion	Caracter.	Respuesta para el menú.
camino1	Arreglo caracter.	Ubicación de las figuras.
boton	Entero.	Monitoreo del ratón.
membresía	Arreglo a caracter no signado.	Resultados de la operación pseudo-borrosa.
pixel_central	Caracter no signado.	Pixel central de la mascara

		en la tecnica de comparación.
radio	Caracter no signado.	Radio de comparación en la técnica de comparación.
struct centroides { int x,y; } caux,c1,c2	Tipo Struct centroides.	Centroides actual, anterior y de respaldo.
v1x,v1y	Enteros.	Estimación de la velocidad.
datos	Apuntador a archivo.	Archivo de datos.
camara_x_aux, camara_y_aux	Enteros.	Respaldos de la posición inicial de la cámara.
inc_x,inc_y	Enteros.	Incrementos de control dados por el CLB.
repite_t	Caracter.	'S' indica que el objeto ha comenzado a repetir la trayectoria.

Tabla A.3.1. metodo.tecnica

Tabla A.3.2

Variable	Tipo	Descripción
wx,wy	Flotante.	Referencias en "x", "y" para el CLB.
ke	Flotante.	Constante de normalización para el error.
kd	Flotante.	Constante de normalización para la derivada del error.
kdu	Flotante.	Constante de normalización para la salida de control.
kv	Flotante.	Constante de normalización para la velocidad.
vel1x, vel1y	Flotante.	Velocidades.
u	Flotante.	DOF (degree of firing), son nueve reglas.
fi	Flotantes.	Funciones de salida por regla, (singleton), son nueve reglas.
F	Flotantes.	Función de salida global.
i	Entero.	Contador.

ganx,gany	Entero.	Registros de las ganancias a aplicar.

Tabla A.3.2. metodo.predictor_difuso

Tabla A.3.3

Variable	Tipo	Descripción
kv	Flotante.	Constante de normalización para la variable "velocidad".
kdv	Flotante.	Constante de normalización para la derivada de la velocidad: "aceleracion".
u	Flotantes.	DOF (degree of firing), son nueve reglas.
fi	Flotantes.	Funciones de salida por regla, (singleton), son nueve reglas.
F	Flotantes.	Función de salida global.
i	Entero.	Contador.

Tabla A.3.3. metodo.ganancia_dinamica_FLC

Tabla A.4

Variable o Función	Tipo	Descripción
inicializando(void)	---	Constructor.
inicializa(void)	Void.	Inicializa gráficos.
ini_pal(int n, unsigned char *p, int fondo)	Void.	Inicializa paleta de colores a tono de grises.
inicio(int modo_deseado)	Void.	Inicializa modo de resolución gráfica deseado.

Tabla A.4. Clase inicializando.

Tabla A.4.1

Variable	Tipo	Descripción
paleta	Apuntador a Archivo.	Apunta al archivo que guarda la paleta de colores.
pal	Arreglo caracter no signado.	Almacena la paleta de colores leída del archivo.

Tabla A.4.1. inicializando.inicio

Tabla A.5

Variable o Función	Tipo	Descripción
despliega(char *camino,unsigned char tam,char tipo,int x, int y);	Void.	Despliega el icono localizado en "camino" en las coordenadas "x" e "y", cuyo tamaño es "tam" y "tipo" indica 16×16 o 32×32 pixeles.

Tabla A.5. Clase Icono.

Tabla A.5.1

Variable	Tipo	Descripción
i,j	Enteros.	Contadores.
bufer	Arreglo caracter no signado.	Arreglo para almacenar el icono leído del archivo.
ent	Apuntador a archivo.	Apuntador al archivo que almacena el icono a desplegar.

Tabla A.5.1 Icono.despliega

CÓDIGO FUENTE.

```
/*****************************************************************************************/
/*----------------------Autor: Jorge Valeriano Assem----------------------*/
/*-------------------------------SIMULADOR C++ ----------------------------*/

/*Simulador para el seguimiento de un objeto en movimiento, mediante una camara
  con resolución de 128*128 pixeles, empleando Procesamiento Digital de Imágenes
  y Control Lógico Borroso (CLB).
  Incluye las técnicas de Procesamiento de Imágenes.
  *Filtrado espacial Laplaciano "L"
  *Filtrado espacial con Pseudo-Lógica Borrosa "D"
  *Filtrado espacial por Comparación de Mínima Distancia  "C"
   Esta versión usa un CLB para el seguimiento del objeto con tres funciones de
   membresía por variable ("error" y "variación error"), ESTA VERSIÓN incluye
   además la variable de entrada "velocidad" al CLB, la cual utiliza funciones de
   membresía exponenciales. La base de conocimiento del CLB tiene nueve reglas
   La GANANCIA del CLB SE CALCULA DINÁMICAMENTE, mediante otro
   CLB que emplea tres funciones de membresía y nueve reglas. Además agrega
   un predictor de velocidad
   Todos los parámetros de normalización y reglas se encuentran sintonizadas.
   El desplazamiento del objeto es evaluado sobre tres trayectorias fijas, permitiendo
   una comparación directa para cualquier variación en los parámetros o técnicas
   utilizadas. */

/*****************************************************************************************/

/*NOTAS:    * Compilar en modelo Huge
             * Tipo de Código 286 con coprocesador 80286/387 en opciones de
               compilación
             * En opciones debuger seleccionar un heap de 128 K
             * Instalar un driver del mouse con capacidad para despliegue
               SVGA
  FIN NOTAS */

/*****************************************************************************************/
```

```
/*----------Inclusión de archivos headers----------*/
#include <conio.h>
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <iostream.h>
#include <alloc h>
#include <math.h>
#include <string.h>
#include "mouse.cpp"                    //funciones para el mouse
#include "mouse h"
#include "cursor.h"

/*----------Declaración de variables globales----------*/
int modo;                               //modo de video en uso

/*----------Declaración de clases----------*/

/*Esta clase define la camara de video*/
class camara {
    int posx, posy;                     //lugar donde se visualiza la camara
    unsigned char huge * imagen;        //apuntador a la imagen capturada por la camara
  public:
    int pos_act_ren;                    //posición actual renglones de la camara en pantalla
    int pos_act_col;                    //posición actual columnas de la camara en pantalla
    unsigned char num_ren;              //numero de renglones de la camara (alto)
    unsigned char num_col;              //numero de columnas de la camara (ancho)
    camara(int nr, int nc);             //constructor
    ~camara(void);                      //destructor
    void prepara(unsigned char nr, unsigned char nc,int actual_x, int actual_y);  //prepara la camara
    unsigned char huge * captura(void);                                           //captura la imagen
};

/*Esta clase define el objeto que estará en movimiento*/
class objeto {
    unsigned char num_ren;              //numero de renglones de la imagen
    unsigned char num_col;              //numero de columnas de la imagen
    unsigned char huge * imagen;        //apuntador a la imagen

  public:
    int pos_act_ren;                    //posición actual renglones de la imagen en pantalla
    int pos_act_col;                    //posición actual columnas de la imagen en pantalla
    unsigned char huge *x,huge *y;      //coordenadas de la trayectoria columna, renglon
    objeto(void);                       //constructor
    ~objeto(void);                      //destructor
    void lee(char * camino, unsigned char nr, unsigned char nc);          //lee imagen de un archivo
    void mover(int renglon, int columna,char bandera);                    //mueve imagen a nuevo renglon columna
    void poner(int renglon, int columna);                                 //coloca imagen en renglon columna
    unsigned int elemento(unsigned char huge * arreglo,unsigned int elemento, unsigned int valor, char acceso),  //lee o
                                                                          //escribe un elemento en x o y
    void trayectoria(void);                                               //traza la trayectoria
    void linea(int x1, int y1, int x2, int y2, int *indice,unsigned char huge *x,unsigned char huge *y,int color); //traza una recta
};                                                                        //entre los puntos


//Esta clase define los métodos y el CLB en base a los cuales se procesa la imagen
class metodo {
  float e1x,e2x;                        //error actual(2) y anterior(1) en x (columnas) para el CLB
  float e1y,e2y;                        //error actual(2) y anterior(1) en y (renglones) para el CLB
  float dex,dey;                        //derivadas (variaciones) del error para el CLB
  float vel1x,vel1y;                    //velocidad para el CLB
  float xs;                             //variación de la simetría de las funciones de membresía (0,1)
  float xv;                             //variación de la forma de las funciones positivo y negativo
para
                                        //la variable de velocidad del CLB (-1,1)
```

```
  float alfa,beta;                                          //parámetro para la exponencial
  float corrimiento;                                        //parámetro para la exponencial
  float beta_ganancia;                                      //beta para la ganancia dinámica
  float velocidadx,aceleracionx, velocidady, aceleraciony;  //datos para el CLB que calcula ganancias
  public.
    metodo(void);                                           //constructor
    void tecnica(char *camino,int imagen_nr,int imagen_nc,int camara_nr, int camara_nc, char tec),   //aplica la tecnica
                                                                                                    //especificada en "tec"
    float bajo(unsigned char x),                            //función de membresía 1
    float medio(unsigned char x);                           //función de membresía 2
    float alto(unsigned char x),                            //función de membresía 3
    unsigned char OR_difusa(float x, float y, float z),     //operador borroso
    void predictor_difuso(int xo, int yo,int v1x, int v1y, int *dux, int *duy),   //predictor borroso
    float negativo(float x);                                //función de membresía
    float zero(float x);                                    //función de membresía
    float positivo(float x),                                //función de membresía
    float min_and(float a, float b, float c);               //AND borrosa, selecciona el mínimo
    float max_or(float a, float b, float c);                //OR borrosa
    float positivo_vel(float x);                            //función de membresía para la velocidad
    float negativo_vel(float x);                            //función de membresía para la velocidad
    float positivo_vel_e(float x);                          //función de membresía para la velocidad
    float negativo_vel_e(float x),                          //función de membresía para la velocidad
    void ganancia_dinamica_FLC(int v1x, int v1y,int *gananciax, int *ganancia);   //devuelve una ganancia dinámica
                                                                                  //calculada con un CLB
};

/*Esta clase se encarga de inicializar el modo, y la paleta de colores a tono de grises*/
class inicializando {
  public.
    inicializando(void);                                    //constructor
    void inicializa(void);                                  //inicializa gráficos
    void ini_pal(int n, unsigned char *p, int fondo);       //inicializa paleta a tono de grises
    void inicio(int modo_deseado);                          //inicializa modo deseado
};

/*Esta clase visualiza iconos*/
class Icono {
  public·
    void despliega(char *camino,unsigned char tam,char tipo,int x, int y);
};

/*----------Definición de funciones clase Icono----------*/

//Esta función recibe la ruta donde se encuentra el icono en formato raw, el tamaño por ejemplo
//16*16, (16) o 32*32 (32), si tipo es 'D' despliega, si tipo es 'O' lo oculta , x e y es la esquina
//superior izquierda donde se desplegara

void Icono..despliega(char *camino,unsigned char tam,char tipo,int x, int y) {
  int i,j,
  unsigned char bufer[1024];
  FILE *ent;
  ent=fopen(camino,"rb");
  if(ent==NULL) {
    cout<<"No existe el archivo pulse una tecla  . ";
    getch();
    exit(1);
  }
  fread(bufer,tam,tam,ent),                 //lee icono
  fclose(ent),
  for(i=0, i < tam,i++)                     //despliega u oculta icono
    for(j=0, j < tam;j++)
          putpixel(x+j,y+i,((((tipo=='D')?bufer[i*tam+j]:0)*-1+255+1)>255)?255.(((tipo=='D')?bufer[i*tam+j] 0)*-1+255+1));
}

/*----------Definición de funciones de la clase inicializando----------*/

inicializando::inicializando(void){
```

```
  modo=3;
}

int huge DetectVGA256(void){
  return modo;
}

void inicializando::inicializa(void){
  int Gd,Gm;
  Gd=installuserdriver("Svga256", DetectVGA256);
  Gd=DETECT;
  initgraph(&Gd,&Gm,"");
}

//inicia la paleta de la siguiente manera: posición 0 color de fondo (negro 0),
//posición 1 color blanco(255), posición 2... tonos de gris hasta la posición
//255 que es color negro

void inicializando::ini_pal(int n, unsigned char *p, int fondo){
   int i;
   union REGS r;
   unsigned char *aux;
   aux=p;
   p++;p++;p++;                    //se salta el negro de la paleta
   outp(0x3c6,0xff);               //es decir en la posición 255 estará el 1 casi negro
   for(i=n-1; i>0;i--){            //paleta invertida parea hacer visible el mouse.
     outp(0x3c8,i);                //para obtener despliegues correctos aplicar la
     outp(0x3c9,(*p++) >> 2),      //transformación y=-x+255+1 sólo a los despliegues
     outp(0x3c9,(*p++) >> 2);      //verificar el limite de 255. Elemento 0 color del fondo
     outp(0x3c9,(*p++) >> 2);      //0 negro, 1 blanco 2 3.... tonos de gris decrecientes
   }
   outp(0x3c6,0xff);               //pone el color de fondo en la
   outp(0x3c8,0);                  //entrada cero de la paleta con el
   p=aux+fondo*3,                  //valor pasado en "fondo"
   outp(0x3c9,*p++ >> 2);
   outp(0x3c9,*p++ >> 2);
   outp(0x3c9,*p++ >> 2);
}

void inicializando::inicio(int modo_deseado) {
   FILE *paleta;
   unsigned char pal[768];
   modo=modo_deseado;
   inicializa();
   paleta=fopen("paleta.pal","rb");
   fread(pal,sizeof(unsigned char),768,paleta);
   fclose(paleta);
   ini_pal(256,pal,0);             //fondo negro
   setcolor(1);                    //color blanco para el trazado de líneas
   setfillstyle(SOLID_FILL,255);   //relleno casi negro que desde el punto
                                   //lógico será el color de fondo
   floodfill(0,0,1);
}

/*------------Definición de funciones de la clase objeto----------------*/

//Constructor
objeto::objeto(void) {
   num_ren=num_col=0;
   pos_act_ren=pos_act_col=0;
   imagen=NULL,
   x=(unsigned char huge *)farmalloc(10000L);
   y=(unsigned char huge *)farmalloc(10000L);
   if(x==NULL || y==NULL) {
     cout<<"No hay memoria pulse enter ...";
     getch();
     exit(1);
```

```
  }
}

//Destructor
objeto ~objeto(void) {
  farfree(imagen);
  farfree(x);
  farfree(y);
}

//lee la imagen del archivo especificada en camino con nr  y nc renglones
//y columnas respectivamente, devolviendo un apuntador a la imagen
//que se le aplica previamente la transformación y=-x+255+1 para hacerla
//compatible con la paleta de grises establecida

void objeto::lee(char * camino, unsigned char nr, unsigned char nc) {
  FILE *tempo;
  unsigned char huge *auxiliar;                    //apuntador para el desplazamiento
  tempo=fopen(camino,"rb");
  if(imagen==NULL)
    imagen=(unsigned char huge *)farmalloc(1L*nr*nc+10L);    //memoria para la imagen
  if(tempo==NULL || imagen==NULL ) {
    printf("\nNo hay memoria suficiente o archivo no encontrado pulse <enter>...");
    getch();
    exit(1);
  }
  num_ren=nr; num_col=nc;
  auxiliar=imagen,                                 //dirección del primer elemento
  *auxiliar=nc-1;auxiliar++;*auxiliar=0;auxiliar++;
  *auxiliar=nr-1;auxiliar++;*auxiliar=0;           //prepara dimensiones para la
  auxiliar++;                                      //la función  putimage
  for(int i=0,i<nr;i++)
    for(int j=0;j<nc,j++) {
          fread(auxiliar,1,1,tempo);               //transformación
          if(*auxiliar!=0)
            *auxiliar=-1*(*auxiliar)+255+1;
          else
            *auxiliar=255;
          auxiliar++;
    }
  fclose(tempo);
}

//coloca la imagen en renglon columna

void objeto::poner(int renglon, int columna) {
  putimage(columna,renglon,imagen,COPY_PUT);       //coloca imagen
}

//mueve la imagen de pos_act_ren pos_act_col al nuevo renglon columna
//borrando la imagen anterior, evitando el parpadeo

void objeto::mover(int renglon, int columna, char bandera){
  unsigned char huge * prueba; unsigned char huge *apunta;
  prueba=(unsigned char huge *)farmalloc(1L*num_ren*num_col+10L),
  if(prueba==NULL) {
    printf("\nNo hay memoria suficiente pulse <enter>...");
    getch();
    exit(1);
  }
  apunta=prueba;
  if(bandera=='N') {                               //eliminación de partes verticales
    int ancho=abs(pos_act_col-columna),
    *apunta=(unsigned char)(ancho);apunta++;*apunta=0;apunta++;*apunta=(unsigned char)(num_ren-
                                                                    1);apunta++;*apunta=0;apunta++,
    for(int i=0,i<num_ren*(ancho+2),i++){          //copia ceros
          *apunta=255,                             //color de fondo casi negro, para evitar bordes falsos
```

```
            apunta++;
        }
        if(columna>=pos_act_col && renglon>=pos_act_ren)
            putimage(pos_act_col,pos_act_ren,prueba,COPY_PUT);
        if(columna>=pos_act_col && renglon<=pos_act_ren)
            putimage(pos_act_col,pos_act_ren,prueba,COPY_PUT),
        if(columna<=pos_act_col && renglon<=pos_act_ren)
            putimage(columna+num_col,pos_act_ren,prueba,COPY_PUT);
        if(columna<=pos_act_col && renglon>=pos_act_ren)
            putimage(columna+num_col,pos_act_ren,prueba,COPY_PUT);

        //eliminación de partes horizontales

        apunta=prueba;
        int alto=(unsigned char)(abs(pos_act_ren-renglon));
        *apunta=(unsigned char)(num_col-1);apunta++;*apunta=0;apunta++,*apunta=(unsigned
                                                    char)(alto);apunta++;*apunta=0;apunta++;
        for(i=0;i<(alto+2)*num_col;i++){                    //copia ceros
            *apunta=255;                                    //color de fondo casi negro evitando bordes falsos
            apunta++;
        }
        if(columna>=pos_act_col && renglon>=pos_act_ren)
            putimage(pos_act_col,pos_act_ren,prueba,COPY_PUT);
        if(columna>=pos_act_col && renglon<=pos_act_ren)
            putimage(pos_act_col,renglon+num_ren,prueba,COPY_PUT);
        if(columna<=pos_act_col && renglon<=pos_act_ren)
            putimage(pos_act_col,renglon+num_ren,prueba,COPY_PUT);
        if(columna<=pos_act_col && renglon>=pos_act_ren)
            putimage(pos_act_col,pos_act_ren,prueba,COPY_PUT);
        putimage(columna,renglon,imagen,COPY_PUT),  //coloca imagen
    }
    else {
        apunta=prueba;
        *apunta=(unsigned char)(num_col-1);apunta++;*apunta=0;apunta++;*apunta=(unsigned char)(num_ren-
                                                                    1);apunta++;*apunta=0;apunta++;
        for(int i=0;i<num_ren*num_col;i++){                 //copia ceros
            *apunta=255;                                    //color de fondo casi negro para evitar bordes falsos
            apunta++;
        }
        putimage(pos_act_col,pos_act_ren,prueba,COPY_PUT);      //borra imagen
        putimage(columna,renglon,imagen,COPY_PUT);              //coloca imagen
    }
    farfree(prueba),
    pos_act_col=columna; pos_act_ren=renglon;
}

//Esta función almacena números de 10 bits en los arreglos x e y de la clase.
//Recibe. el arreglo, elemento del mismo, valor del elemento y accesso='L' para
//lectura y acceso='E' para escritura. Esto con el fin de optimizar espacio.

unsigned int objeto::elemento(unsigned char huge * arreglo,unsigned int elemento, unsigned int valor, char acceso) {
    unsigned long bit_ini,bit_fin;                          //bits entre los que se halla el dato
    unsigned int byte_ini,byte_fin;                         //bytes que contienen la información
    unsigned int bit_fin_byte_fin,                          //bits dentro de los que se halla el dato  en
    unsigned int temporal,mascara,desplaza;                 //los bytes que contienen la información
    unsigned int datos;                                     //contiene los bytes de interés en formato entero
    unsigned int valor_aux;                                 //auxiliar del valor que se pasa como parámetro
    bit_fin=(elemento+1)*10L-1L;                            //comienza en cero
    bit_ini=bit_fin-10L+1L;
    byte_ini=(unsigned int)(bit_ini/8);
    byte_fin=(unsigned int)(bit_fin/8);
    bit_fin_byte_fin=(unsigned int)(bit_fin%8);
    desplaza=7-bit_fin_byte_fin;
    mascara=0x03ff;                                         //mascara para bajar la información
    mascara=mascara<<desplaza;                              //colocando mascara en la posición correcta dentro del entero
    datos=(*(arreglo+byte_fin))+256*(*(arreglo+byte_ini));  //valor de los bytes bajados
    if(acceso=='L') {                                       //efectúa la lectura,
```

```
    valor_aux=datos & mascara,
    valor_aux=valor_aux >> desplaza;
  }
  if(acceso=='E') {                                    //efectúa la escritura
    valor_aux=valor,
    valor_aux=valor_aux << desplaza;
    mascara=~mascara,
    temporal=mascara & datos;
    datos=temporal | valor_aux;
    *(arreglo+byte_ini)=(unsigned char)(datos/256);                    //byte alto
    *(arreglo+byte_fin)=(unsigned char)(datos-256*(*(arreglo+byte_ini)));    //byte bajo
  }
  return(valor_aux);
}

//Esta función obtiene la trayectoria deseada del objeto

void objeto.:trayectoria(void) {
  int xs,ys,xa=10,ya=300,                    //posiciones actual y anterior del pixel señalado
  int indice=0;                              //índice al elemento siguiente a accesar.
  int contador,
  int num_c=100;                             //numero de puntos a graficar para trayectoria constante
  int num_l=80;                              //numero de puntos a graficar para trayectoria lineal
  int a=4,b=0;                               //constantes para los incrementos en x e y
  float alfa=0.2, beta=0.0;                  //parámetros de las líneas en x e y
  char trayectoria='C';                      //tipo de trayectoria 'C'= Constante, 'L'=lineal
                                             //creciente, 'l'=lineal decreciente

  elemento(x,indice,xa,'E');
  elemento(y,indice,ya,'E'),
  indice++;
  if(trayectoria=='C') {                     //trayectoria constante
    for(contador=0,contador<num_c,contador++) {
        xs=xa+a; ys=ya+b;                    //calculo del siguiente punto
        elemento(x,indice,xs,'E');
        elemento(y,indice,ys,'E');
        indice++;
        xa=xs; ya=ys,
    }
    elemento(x,indice,1023,'E');             //fin de trayectoria;
    elemento(y,indice,1023,'E');
    for(int i=0;i<indice,i++)                //pinta la trayectoria
        putpixel(elemento(x,i,0,'L'),elemento(y,i,0,'L'),1),
    getch();
    for(i=0;i<indice;i++)                                  //repinta la trayectoria en color de fondo para no
        putpixel(elemento(x,i,0,'L'),elemento(y,i,0,'L'),255);    //afectar el calculo del centroide
  }
  if(trayectoria=='L') {                                   //trayectoria lineal creciente
    for(contador=0;contador<num_l;contador++) {
        xs=(int)((float)xa+alfa*(contador+1)),             //calculo del siguiente punto
        ys=(int)((float)ya+beta*(contador+1)),             //calculo del siguiente punto
        elemento(x,indice,xs,'E');
        elemento(y,indice,ys,'E'),
        indice++;
        xa=xs; ya=ys;
    }
    elemento(x,indice,1023,'E');             //fin de trayectoria;
    elemento(y,indice,1023,'E');
    for(int i=0;i<indice,i++)                //pinta la trayectoria
        putpixel(elemento(x,i,0,'L'),elemento(y,i,0,'L'),1),
    getch();
    for(i=0;i<indice;i++)                                  //repinta la trayectoria en color de fondo para no
        putpixel(elemento(x,i,0,'L'),elemento(y,i,0,'L'),255);    //afectar el calculo del centroide
  }
  if(trayectoria=='l') {                                   //trayectoria lineal decreciente
    for(contador=num_l-1;contador>=0,contador--) {
        xs=(int)((float)xa+alfa*(contador+1));             //calculo del siguiente punto
```

```
            ys=(int)((float)ya+beta*(contador+1));          //calculo del siguiente punto
            elemento(x,indice,xs,'E');
            elemento(y,indice,ys,'E');
            indice++;
            xa=xs; ya=ys;
    }
    elemento(x,indice,1023,'E');                            //fin de trayectoria;
    elemento(y,indice,1023,'E');
    for(int i=0;i<indice;i++)                               //pinta la trayectoria
            putpixel(elemento(x,i,0,'L'),elemento(y,i,0,'L'),1);
    getch();
    for(i=0;i<indice;i++)                                   //repinta la trayectoria en color de fondo para no
            putpixel(elemento(x,i,0,'L'),elemento(y,i,0,'L'),255);  //afectar el calculo del centroide
  }
}

/* Descomentar para seleccionar una trayectoria aleatoria con el mouse */
  //char bandera='S';             //'N' indica que la coordenada no es valida, 'S' lo contrario


  //mouse.Move(px,py);
  /*int boton;
  do {                          //espera se pulse boton izquierdo del mouse
    mouse.GetEvent();
    boton=mouse.Pressed(0);
  } while(boton==0);
  mouse.Position();
  pxa=mouse.xPos(),
  pya=mouse.yPos();
  do {
    mouse.Position();
    px=mouse.xPos();
    py=mouse.yPos();
    bandera=((pxa>(getmaxx()-128-1)) | (pxa<4) | (bandera=='N'))?'N':'S';
    bandera=((pya>(getmaxy()-128-1)) | (pya<136) | (bandera=='N'))?'N':'S';
    bandera=((px>(getmaxx()-128-1)) | (px<4) | (bandera=='N'))?'N':'S';
    bandera=((py>(getmaxy()-128-1)) | (py<136) | (bandera=='N'))?'N':'S';
    if((bandera=='S')&&((px!=pxa)||(py!=pya))) {
            mouse.Hide();
            linea(pxa-1,pya-1,px-1,py-1,&indice,x,y,1);
            mouse.Show();
            //putpixel(pxa-1,pya-1,1);          //dibuja pixel en posición del mouse en color blanco
            //elemento(x,indice,pxa-1,'E');
            //elemento(y,indice,pya-1,'E');
            //indice++;
            pxa=px; pya=py,
    }
    else { pxa=px; pya=py;}
    bandera='S';
  } while(!kbhit());
  elemento(x,indice,1023,'E');                      //fin de trayectoria;
  elemento(y,indice,1023,'E');
  for(int i=0;i<indice;i++)                         //repinta la trayectoria en color de fondo para no
    putpixel(elemento(x,i,0,'L'),elemento(y,i,0,'L'),255);  //afectar el calculo del centroide
} */

//Traza una linea recta entre (x1,y1) y (x2,y2),con color, actualizando el indice de elementos,
//y la trayectoria apuntada por *x , *y
void objeto::linea(int x1, int y1, int x2, int y2, int *indice,unsigned char huge *x,unsigned char huge *y,int color){
  int xi,yi,i,j,k;                                  //punto inicial y contadores
  float m;                                          //pendiente
  char bandera;                                     //indica como se calcula la recta y=f(x) o x=f(y)
  if((x2-x1)!=0) {                                  //recta con pendiente diferente a 90 grados
    m=(1.0*(y2-y1))/(1.0*(x2-x1));                  //pendiente
    xi=x1; yi=y1;                                   //punto inicial
    i=abs(x2-x1);//+1;              //puntos a graficar incluyendo sólo el extremo inicial, pero no el final
    k=abs(y2-y1);//+1;              //puntos a graficar incluyendo sólo el extremo inicial, pero no el final
```

```
    if(i>=k) {
          bandera='S';                    //se usa y=f(x)
    }
    else {
          i=k; bandera='N';               //se usa x=f(y);
    }
    for(j=0,j<i;j++) {                    //graficación de los puntos
          putpixel(xi,yi,color);          //pixel en color blanco
          elemento(x,*indice,xi,'E'),     //guarda el punto en la trayectoria
          elemento(y,*indice,yi,'E'),
          (*indice)++;
          if(bandera=='S') {
            xi=(x1<x2)?xi+1:xi-1,         //x creciente o decreciente
            yi=(int)(m*(xi-x1)+y1);
          }
          else {
            yi=(y1<y2)?yi+1:yi-1;         //x creciente o decreciente
            xi=(int)((yi-y1)/m+x1);
          }
    }
  }
  else {                                  //pendiente de 90 grados
    xi=x1; yi=y1,                         //punto de inicio
    i=abs(y2-y1),//+1;                    //incluye sólo el punto inicial, pero no el final
    for(j=0;j<i,j++) {                    //graficación de los puntos
          putpixel(xi,yi,color);          //pixel en color blanco
          elemento(x,*indice,xi,'E');     //guarda el punto en la trayectoria
          elemento(y,*indice,yi,'E'),
          (*indice)++,
          yi=(y1<y2)?yi+1:yi-1;           // y creciente o decreciente
    }
  }
}


/*----------Definición de funciones de la clase camara----------*/

//constructor, recibe el numero de columnas y renglones de la camara

camara::camara(int nr, int nc) {
  imagen=(unsigned char huge *)farmalloc((1L*nr*nc+10L));
  if(imagen==NULL) {
    printf("\nNo hay memoria suficiente pulse <enter> ..");
    getch(),
    exit(1);
  }
  posx=posy=0;
  pos_act_ren=pos_act_col=0;
  num_ren=num_col=0;
}

//destructor

camara::~camara(void) {
  farfree(imagen),
}

//Prepara en pantalla la camara, dibujando el recuadro en x y, con nr renglones
//y nc columnas, las posiciones actuales de inicio de captura de camara se
//dan en actual_x(columnas) y actual_y(renglones). También dibuja iconos
//de menú

void camara::prepara(unsigned char nr, unsigned char nc,int actual_x, int actual_y){
  char icono_camara[50]={"i_cam.gry"};          //archivo con icono de la camara
  char icono_salir[50]={"i_salir.gry"};         //archivo con icono salir
  char icono_trayectoria[50]={"i_tray.gry"};    //archivo con icono de nueva trayectoria
  char icono_t_objeto[50]={"i_t_obj.gry"};      //archivo con icono de tiempo de movimiento del objeto
```

```
 char icono_t_camara[50]={"i_t_cam.gry"};          //archivo con icono de tiempo de muestreo de la camara
 char icono_figura1[50]={"i_fig1.gry"};            //archivo con icono para fig 1
 char icono_figura2[50]={"i_fig2 gry"};            //archivo con icono para fig2
 char icono_figura3[50]={"i_fig3.gry"};            //archivo con icono para fig3
 char icono_figura4[50]={"i_fig4.gry"};            //archivo con icono para fig4
 Icono icono;
 posx=getmaxx()-128-1; posy=1;                     //posición de la camara, ángulo superior izquierdo
 num_ren=nr; num_col=nc;
 rectangle(posx-1,posy-1,posx+num_col+1,posy+num_ren+1);      //cuadro camara
 pos_act_ren=actual_y; pos_act_col=actual_x;
 icono.despliega(icono_camara,32,'D',getmaxx()-180+5,48);     //despliega icono de la camara
 line(getmaxx()-180+5+32,69,posx-1,69);                       //cable de conexión
 rectangle(0,0,134,130);                                      //cuadro para los iconos
 rectangle(136,0,getmaxx()-180,130);                          //cuadro para los mensajes
 rectangle(0,133,getmaxx(),getmaxy());                        //cuadro para el despliegue
 icono.despliega(icono_t_objeto,32,'D',3,3);                  //despliega icono· periodo movimiento del objeto
 icono.despliega(icono_t_camara,32,'D',3,37);                 //despliega icono. periodo muestreo de la camara
 icono.despliega(icono_trayectoria,32,'D',3,71);              //despliega opcion de nueva trayectoria
 icono.despliega(icono_figura1,32,'D',45,3),                  //despliega icono para figura 1
 icono.despliega(icono_figura2,32,'D',45,37);                 //despliega icono para figura 2
 icono.despliega(icono_figura3,32,'D',45,71);                 //despliega icono para figura 3
 icono.despliega(icono_figura4,32,'D',87,3);                  //despliega icono para figura 4
 icono.despliega(icono_salir,32,'D',87,37);                   //despliega icono para salir
}

//captura la imagen que se obtenga desde pos_act_ren y pos_act_col  de la
//posición de la camara y devuelve un apuntador a la imagen

unsigned char huge * camara:.captura(void) {
  getimage(pos_act_col,pos_act_ren,pos_act_col+num_col-1,pos_act_ren+num_ren-1,imagen);
  putimage(posx,posy,imagen,COPY_PUT);
  return(imagen);
}

/*----------Definición de funciones de la clase método----------*/

//Constructor

metodo::metodo(void){
  e1x=e2x=e1y=e2y=dex=dey=vel1x=vel1y=0;              //inicialización para CLB
  velocidadx=aceleracionx=velocidady=aceleraciony=0;  //inicialización para el CLB de ganancia
}

//Funciones de membresía y operadores borrosos para la tecnica pseudo-borrosa de procesamiento de la imagen

float metodo::bajo(unsigned char x) {                 //función de membresía 1
  return((x<=127)?-0.007874015748031*(float)x+1:0);
}

float metodo::medio(unsigned char x) {                //función de membresía 2
  return(x>=63 && x<=191)?((x>=63 && x<=127)?0.01587301587302*(float)x-1:-0.01587301587302*(float)x+3):0);
}

float metodo::alto(unsigned char x) {                 //función de membresía 3
  return((x>=127)?0.007874015748031*(float)x-1:0);
}

unsigned char metodo::OR_difusa(float x, float y, float z) {  //operador borroso OR
  if(x>=y && x>=z)  return(1);
  else if(y>=x && y>=z) return(2);
          else return(3),
}

// función de merecía para el FLC

float metodo.:negativo(float x) {
  if(x<=-xs)
```

```
    return(1 0);
  else
    if((-xs<x) && (x<=0))
          return((-1.0/xs)*x),
  return(0);
}

// función de merecía para el FLC

float metodo::zero(float x) {
  if((-xs<=x) && (x<0))
    return((1 0/xs)*x+1);
  else
    if((0<=x) && (x<=xs))
          return((-1.0/xs)*x+1);
  return(0);
}

// función de merecía para el FLC

float metodo::positivo(float x) {
  if((0<=x) && (x<xs))
    return((1 0/xs)*x);
  else
    if(x>=xs)
          return(1.0),
  return(0),
}

// función de membresía para el FLC

float metodo::positivo_vel(float x) {                    //función de membresía para la velocidad
  if((-1.0<=x) && (x<=-xv))
    return((1.0/(1 0-xv))*(x+1.0)),
  else
    if(x>-xv)
          return(1.0);
  return(0);
}

//función de membresía para el FLC

float metodo::negativo_vel(float x) {                    //función de membresía para la velocidad
  if((xv<=x) && (x<=1 0))
    return((-1.0/(1 0-xv))*(x-1.0)),
  else
    if(x<xv)
          return(1.0);
  return(0),
}

// función de membresía para el FLC

float metodo::positivo_vel_e(float x) {                  //función de membresía para la velocidad
  if((-1 0+corrimiento<=x) && (x<=corrimiento))
    return((float)(alfa*exp(beta*(x-corrimiento))));
  else
    if(x>corrimiento)
          return(1 0),
  return(0);
}

//función de membresía para el FLC

float metodo::negativo_vel_e(float x) {                  //función de membresía para la velocidad
  if((-corrimiento<=x) && (x<=1.0-corrimiento))
    return((float)(alfa*exp(-beta*(x+corrimiento)))),
```

```
  else
    if(x<-corrimiento)
          return(1.0);
  return(0);
}

// Esta función efectúa una  AND borrosa para el CLB, selecciona el mínimo

float metodo:.min_and(float a, float b, float c) {
  if((a<=b) && (a<=c))
    return(a);
  else
    if((b<=a) && (b<=c))
          return(b);
    else
          return(c);
}

// Esta función efectúa  una OR borrosa para el CLB, selecciona el máximo

float metodo::max_or(float a, float b, float c) {
  if((a>=b) && (a>=c))
    return(a);
  else
    if((b>=a) && (b>=c))
          return(b);
    else
          return(c);
}

//devuelve una ganancia dinámica calculada con un CLB el cual posee tres funciones de
//membresía y nueve reglas.

void metodo::ganancia_dinamica_FLC(int v1x, int v1y,int *gananciax, int *ganancíay){

  float kv=1.0/94.0;                    //constante de normalización para la velocidad
  float kdv=1.0/60.0;                   //constante de normalización para la derivada de la velocidad, aceleracion
  float u[9];                           //DOF (degree of firing), son nueve reglas
  float fi[9][3]={0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0};               //funciones de salida por regla,(singleton), son nueve reglas
  float F[3]={0,0,0};                   //función de salida global
  int i;                                //contador
  // parámetro de simetría de las funciones de membresía
  xs=0.5;
  //Actualizaciones
  aceleracionx=v1x-1.0*velocidadx/kv; aceleraciony=v1y-1.0*velocidady/kv;
  aceleracionx=kdv*aceleracionx; aceleraciony=kdv*aceleraciony;
  velocidadx=kv*v1x; velocidady=kv*v1y;

  //EVALUACIÓN DE LAS REGLAS PARA X
  //Regla 0 Si v(k) es negativo y dv(k) es negativo y kdu es negativo
  u[0]=min_and(negativo(velocidadx),negativo(aceleracionx),1.0);        //DOF
  fi[0][2]=u[0];                                                        //u^d, corte de la función de salida
  //Regla 1 Si v(k) es cero y dv(k) es negativo entonces kdu es negativo
  u[1]=min_and(zero(velocidadx),negativo(aceleracionx),1.0);            //DOF
  fi[1][0]=u[1];                                                        //u^d, corte de la función de salida
  //Regla 2 Si v(k) es positivo y dv(k) es negativo entonces kdu es zero
  u[2]=min_and(positivo(velocidadx),negativo(aceleracionx),1 0);        //DOF
  fi[2][0]=u[2],                                                        //u^d, corte de la función de salida
  //Regla 3 Si v(k) es negativo y dv(k) es cero  entonces kdu es negativo
```

```
u[3]=min_and(negativo(velocidadx),zero(aceleracionx),1.0),          //DOF
fi[3][1]=u[3];                                                      //u^d, corte de la función de salida
//Regla 4 Si v(k) es cero y dv(k) es cero entonces kdu es cero
u[4]=min_and(zero(velocidadx),zero(aceleracionx),1 0),              //DOF
fi[4][1]=u[4];                                                      //u^d, corte de la función de salida
//Regla 5 Si v(k) es positivo y dv(k) es cero entonces kdu es positivo
u[5]=min_and(positivo(velocidadx),zero(aceleracionx),1.0);          //DOF
fi[5][1]=u[5];                                                      //u^d, corte de la función de salida
//Regla 6 Si v(k) es negativo y dv(k) es positivo entonces kdu es cero
u[6]=min_and(negativo(velocidadx),positivo(aceleracionx),1.0),      //DOF
fi[6][0]=u[6],                                                      //u^d, corte de la función de salida
//Regla 7 Si v(k) es cero y dv(k) es positivo entonces kdu es positivo
u[7]=min_and(zero(velocidadx),positivo(aceleracionx),1.0);          //DOF
fi[7][0]=u[7],                                                      //u^d, corte de la función de salida
//Regla 8 Si v(k) es positivo y dv(k) es positivo kdu es positivo
u[8]=min_and(positivo(velocidadx),positivo(aceleracionx),1 0);      //DOF
fi[8][2]=u[8];                                                      //u^d, corte de la función de salida

//Calculo de la función de salida global, máximos
for(i=0;i<9;i++) {
  if(fi[i][0]>=F[0])  F[0]=fi[i][0];
  if(fi[i][1]>=F[1])  F[1]=fi[i][1],
  if(fi[i][2]>=F[2])  F[2]=fi[i][2];
}

//defusificación y obtención de la salida de control desnormalizada
if((F[0]+F[1]+F[2])!=0)
  *gananciax=(int)(((-0.5*F[0]+0*F[1]+0.5*F[2])/(F[0]+F[1]+F[2]))*55.0+95 0);
else                    // evita la división por cero
  *gananciax=145;       //ganancia por default

//EVALUACION DE LAS REGLAS PARA Y
F[0]=F[1]=F[2]=0 ,                          //inicialización de las funciones de salida a cero
//Regla 0 Si u(k) es negativo y du(k) es negativo kdu es negativo
u[0]=min_and(negativo(velocidady),negativo(aceleraciony),1.0);      //DOF
fi[0][2]=u[0],                                                      //u^d, corte de la función de salida
//Regla 1 Si u(k) es cero y du(k) es negativo entonces kdu es negativo
u[1]=min_and(zero(velocidady),negativo(aceleraciony),1 0);          //DOF
fi[1][0]=u[1],                                                      //u^d, corte de la función de salida
//Regla 2 Si u(k) es positivo y du(k) es negativo entonces kdu es zero
u[2]=min_and(positivo(velocidady),negativo(aceleraciony),1.0);      //DOF
fi[2][0]=u[2],                                                      //u^d, corte de la función de salida
//Regla 3 Si u(k) es negativo y du(k) es cero kdu es negativo
u[3]=min_and(negativo(velocidady),zero(aceleraciony),1.0),          //DOF
fi[3][1]=u[3];                                                      //u^d, corte de la función de salida
//Regla 4 Si u(k) es cero y du(k) es cero entonces kdu es cero
u[4]=min_and(zero(velocidady),zero(aceleraciony),1 0),              //DOF
fi[4][1]=u[4],                                                      //u^d, corte de la función de salida
//Regla 5 Si u(k) es positivo y du(k) es cero kdu es positivo
u[5]=min_and(positivo(velocidady),zero(aceleraciony),1.0),          //DOF
fi[5][1]=u[5];                                                      //u^d, corte de la función de salida
//Regla 6 Si u(k) es negativo y du(k) es positivo entonces kdu es cero
u[6]=min_and(negativo(velocidady),positivo(aceleraciony),1 0),      //DOF
fi[6][0]=u[6];                                                      //u^d, corte de la función de salida
//Regla 7 Si u(k) es cero y du(k) es positivo entonces kdu es positivo
u[7]=min_and(zero(velocidady),positivo(aceleraciony),1 0);          //DOF
fi[7][0]=u[7];                                                      //u^d, corte de la función de salida
//Regla 8 Si u(k) es positivo y du(k) es positivo kdu es positivo
u[8]=min_and(positivo(velocidady),positivo(aceleraciony),1 0);      //DOF
fi[8][2]=u[8];                                                      //u^d, corte de la función de salida

//Calculo de la función de salida global
for(i=0,i<9,i++) {
  if(fi[i][0]>=F[0])  F[0]=fi[i][0];
  if(fi[i][1]>=F[1])  F[1]=fi[i][1];
  if(fi[i][2]>=F[2])  F[2]=fi[i][2];
}
```

```
  //defusificación y obtención de la salida de control desnormalizada

  if((F[0]+F[1]+F[2])!=0)
    *gananciay=(int)(((-0.5*F[0]+0*F[1]+0.5*F[2])/(F[0]+F[1]+F[2]))*55.0+95.0);
   else
    *gananciay=140;                              //ganancia por default
}



// Esta función implementa el control de la posición de la camara usando un CLB
// (Controlador Lógico Borroso) como un controlador PI. Recibe las coordenadas
// que describen el centroide de la imagen más reciente (xo,yo) y las utiliza
// para dar tres entradas al CLB que son el error, derivada del error y velocidad del objeto, utiliza
// tres funciones de membresía triangulares para el error y su derivada y dos funciones de
// membresía exponenciales para la velocidad. La base de conocimiento posee nueve
// reglas.
// El parámetro de ganancia es calculado usando otro CLB.
// Devuelve dos incrementos dux y duy que se deben sumar directamente a las
// posiciones actuales de la camara para lograr la corrección.

void metodo::predictor_difuso(int xo, int yo,int v1x, int v1y, int *dux, int *duy) {

  float wx=64.0,wy=64.0;                         //referencias para el control en x e y
  float ke=1.0/74;                               //constante de normalización para el error
  float kd=1.0/135,                              //constante de normalización para la derivada del error
  float kdu=145;                                 //constante de normalización para la salida de control
  float kv=1.0/64;                               //constante de normalización para la velocidad
  float vel1x,vel1y;                             //velocidades en flotante
  float u[9];                                    //DOF (degree of firing), son nueve reglas
  float fi[9][3]={0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0,
                  0,0,0};                        //funciones de salida por regla, (singleton), son nueve reglas
  float F[3]={0,0,0};                            //función de salida global
  int i;                                         //contador
  int ganx,gany;                                 //registros de las ganancias a aplicar

  // obtiene las ganancias a aplicar para cada eje
  ganancia_dinamica_FLC(v1x,v1y,&ganx,&gany),    //devuelve la ganancia a aplicar en x e y usando un CLB

  // parámetro de simetría de las funciones de membresía
  xs=0 5;
  // parámetro de variación para las funciones de membresía (positivo y negativo) de la variable de velocidad
  xv=-0.0;
  // parámetros de las exponenciales
  alfa=1.0;
  beta=8.0;
  corrimiento=0.0;
  beta_ganancia=0.0687;

 //Actualizaciones
  e1x=e2x; e1y=e2y;                              //actualiza error
  e2x=wx-1.0*xo; e2y=wy-1.0*yo;                  //error actual realimentacion negativa
  dex=e2x-(e1x/ke); dey=e2y-(e1y/ke);            //variación del error. Desnormalizamos el error anterior
  e2x=ke*e2x; e2y=ke*e2y;                        //normalizaciones
  dex=kd*dex; dey=kd*dey;
  vel1x=kv*v1x; vel1y=kv*v1y;

  //EVALUACION DE LAS REGLAS PARA X
```

```
//Regla 0 Si e(k) es negativo y de(k) es negativo y v(k) es positivo entonces du es negativo
u[0]=min_and(negativo(e2x),negativo(dex),positivo_vel_e(vel1x));          //DOF
fi[0][0]=u[0];                                                            //u^d, corte de la función de salida
//Regla 1 Si e(k) es cero y de(k) es negativo entonces du es negativo
u[1]=min_and(zero(e2x),negativo(dex),1 0),                                //DOF
fi[1][1/*0*/]=u[1],                                                       //u^d, corte de la función de salida
//Regla 2 Si e(k) es positivo y de(k) es negativo entonces du es zero
u[2]=min_and(positivo(e2x),negativo(dex),1.0);                            //DOF
fi[2][1]=u[2];                                                            //u^d, corte de la función de salida
//Regla 3 Si e(k) es negativo y de(k) es cero y v(k) es positivo entonces du es negativo
u[3]=min_and(negativo(e2x),zero(dex),positivo_vel_e(vel1x));              //DOF
fi[3][0]=u[3];                                                            //u^d, corte de la función de salida
//Regla 4 Si e(k) es cero y de(k) es cero entonces du es cero
u[4]=min_and(zero(e2x),zero(dex),1 0),                                    //DOF
fi[4][1]=u[4];                                                            //u^d, corte de la función de salida
//Regla 5 Si e(k) es positivo y de(k) es cero y v(k) es negativo entonces du es positivo
u[5]=min_and(positivo(e2x),zero(dex),negativo_vel_e(vel1x)),              //DOF
fi[5][2]=u[5];                                                            //u^d, corte de la función de salida
//Regla 6 Si e(k) es negativo y de(k) es positivo entonces du es cero
u[6]=min_and(negativo(e2x),positivo(dex),1.0);                            //DOF
fi[6][1]=u[6],                                                            //u^d, corte de la función de salida
//Regla 7 Si e(k) es cero y de(k) es positivo entonces du es positivo
u[7]=min_and(zero(e2x),positivo(dex),1.0);                                //DOF
fi[7][1/*2*/]=u[7];                                                       //u^d, corte de la función de salida
//Regla 8 Si e(k) es positivo y de(k) es positivo y v(k) es negativo entonces du es positivo
u[8]=min_and(positivo(e2x),positivo(dex),negativo_vel_e(vel1x)),          //DOF
fi[8][2]=u[8],                                                            //u^d, corte de la función de salida

//Calculo de la función de salida global, máximos
for(i=0,i<9;i++) {
  if(fi[i][0]>=F[0])  F[0]=fi[i][0],
  if(fi[i][1]>=F[1])  F[1]=fi[i][1];
  if(fi[i][2]>=F[2])  F[2]=fi[i][2],
}

//defusificación y obtención de la salida de control desnormalizada

kdu=(float)ganx;                    //ganancia calculada para X
if((F[0]+F[1]+F[2])!=0)
  *dux=(int)(((-0 5*F[0]+0*F[1]+0.5*F[2])/(F[0]+F[1]+F[2]))*kdu);
else                                // evita la división por cero
  *dux=0,

//EVALUACION DE LAS REGLAS PARA Y

F[0]=F[1]=F[2]=0 ;                                                        //inicialización de la función resultado a cero
//Regla 0 Si e(k) es negativo y de(k) es negativo y v(k) es positivo entonces du es negativo
u[0]=min_and(negativo(e2y),negativo(dey),positivo_vel_e(vel1y));          //DOF
fi[0][0]=u[0];                                                            //u^d, corte de la función de salida
//Regla 1 Si e(k) es cero y de(k) es negativo entonces du es negativo
u[1]=min_and(zero(e2y),negativo(dey),1.0);                                //DOF
fi[1][1/*0*/]=u[1];                                                       //u^d, corte de la función de salida
//Regla 2 Si e(k) es positivo y de(k) es negativo entonces du es zero
u[2]=min_and(positivo(e2y),negativo(dey),1.0),                            //DOF
fi[2][1]=u[2];                                                            //u^d, corte de la función de salida
//Regla 3 Si e(k) es negativo y de(k) es cero y v(k) es positivo entonces du es negativo
u[3]=min_and(negativo(e2y),zero(dey),positivo_vel_e(vel1y));              //DOF
fi[3][0]=u[3];                                                            //u^d, corte de la función de salida
//Regla 4 Si e(k) es cero y de(k) es cero entonces du es cero
u[4]=min_and(zero(e2y),zero(dey),1.0);                                    //DOF
fi[4][1]=u[4];                                                            //u^d, corte de la función de salida
//Regla 5 Si e(k) es positivo y de(k) es cero y v(k) es negativo entonces du es positivo
u[5]=min_and(positivo(e2y),zero(dey),negativo_vel_e(vel1y)),              //DOF
fi[5][2]=u[5];                                                            //u^d, corte de la función de salida
//Regla 6 Si e(k) es negativo y de(k) es positivo entonces du es cero
u[6]=min_and(negativo(e2y),positivo(dey),1 0);                            //DOF
```

```
  fi[6][1]=u[6];                                                    //u^d, corte de la función de salida
  //Regla 7 Si e(k) es cero y de(k) es positivo entonces du es positivo
  u[7]=min_and(zero(e2y),positivo(dey),1.0);                        //DOF
  fi[7][1/*2*/]=u[7];                                               //u^d, corte de la función de salida
  //Regla 8 Si e(k) es positivo y de(k) es positivo y v(k) es negativo entonces du es positivo
  u[8]=min_and(positivo(e2y),positivo(dey),negativo_vel_e(vel1y));  //DOF
  fi[8][2]=u[8];                                                    //u^d, corte de la función de salida

  //Calculo de la función de salida global
  for(i=0;i<9;i++) {
    if(fi[i][0]>=F[0])  F[0]=fi[i][0];
    if(fi[i][1]>=F[1])  F[1]=fi[i][1];
    if(fi[i][2]>=F[2])  F[2]=fi[i][2];
  }

  //defusificación y obtención de la salida de control desnormalizada
  kdu=(float)gany;                  //ganancia aplicada en Y
  if((F[0]+F[1]+F[2])!=0)
    *duy=(int)(((-0.5*F[0]+0*F[1]+0.5*F[2])/(F[0]+F[1]+F[2]))*kdu);
   else
    *duy=0;                         //evita la división por cero
}


//Esta función aplica el metodo de filtrado en el dominio espacial con base
//en las siguientes técnicas según el parámetro "tec": "L" (Laplace), "F" (pseudo-borrosa),
//"C" (Comparación). Recibe el camino donde está la imagen, renglones y columnas
//de la imagen, renglones y columnas de la camara y la posición x e y de la
//esta. Efectúa el proceso siguiendo toda la trayectoria actual.

void metodo::tecnica(char *camino,int imagen_nr,int imagen_nc,int camara_nr, int camara_nc, char tec){
  float filtro[3][3]={-1.0,-1.0,-1.0,                   //filtro Laplace
                      -1.0,8,-1.0,
                      -1.0,-1.0,-1.0};
  int M=3;                                              //dimensión del filtro.
  unsigned char huge *coor;                             //coordenadas de los bordes [x,y], [1000][2]
  unsigned int nbordes=0;                               //numero de puntos borde
  int A,B;                                              //dimensiones de la imagen de entrada. Renglones y columnas.
  int i,j,k,l;                                          //contadores.
  float suma;                                           //resultado de la convolución
  float suma_a;                                         //resultado de la convolucion anterior
  int y,x;                                              //coordenadas temporales.
  float pixelf;                                         //valor actual del pixel del filtro
  unsigned char pixeli;                                 //valor actual del pixel de la imagen
  unsigned int centroide_int[1][2];                     //centroide entero de la imagen
  float centroide_float[1][2];                          //centroide flotante de la imagen [x,y]
  unsigned int m;                                       //contador
  unsigned char huge * entrada;                         //apuntador a la imagen a procesar
  unsigned char huge * auxiliar;                        //apuntador a la imagen capturada
  unsigned char huge * base;                            //guarda el inicio de auxiliar
  objeto object;                                        //imagen en uso
  camara camare(camara_nr,camara_nc);                   //camara de video
  inicializando comienzo;                               //objeto que inicializa el modo
  int t_objeto=25;                                      //periodo del movimiento del objeto y camara en terceros
  int t_camara=40;                                      //de segundo, es decir en múltiplos de 10mseg
  struct time tp;                                       //obtención del tiempo del sistema
  long int tl_objeto;                                   //tiempos limites para efectuar nuevo movimiento
  long int tl_camara;                                   //del objeto o captura de la camara en centésimas
  long indice;                                          //indice de la trayectoria
  char opcion='z';                                      //respuesta para el menú
  char camino1[50],                                     //ubicación para las diversas figuras
  int boton;                                            //monitoreo del mouse
  unsigned char membresia[5];                           //resultado de la operación pseudo-borrosa
  unsigned char pixel_central;                          //pixel central de la mascara en la tecnica de comparación
  unsigned char radio=10;                               //radio de comparación en la tecnica de comparación
  struct centroides {
    int x,y;
```

```
} caux,c1,c2;                                   //respaldo centroide inicial
int v1x,v1y;                                    //estimación de la velocidad
FILE *datos;                                    //archivo de datos
int camara_x_aux,camara_y_aux;                  //respaldos de la posición inicial de la camara
int inc_x=0,inc_y=0,                            //incrementos de control dados por el CLB

char repite_t='N',                              //'S' indica que el objeto ha comenzado a repetir la trayectoria

// apertura del archivo de datos
datos=fopen("dato3638.txt","wt");

auxiliar=(unsigned char huge *)farmalloc((1L*imagen_nr*imagen_nc+10L));
base=auxiliar;
coor=(unsigned char huge *)farmalloc(1L*10000*2);
if(auxiliar==NULL || coor==NULL) {
  printf("\nNo hay memoria suficiente pulse <enter>. .");
  getch();
  exit(1),
}
comienzo.inicio(modo);                          //inicia resolución de 800*600
object.lee(camino,imagen_nr,imagen_nc);         //lee la imagen de un archivo
//prepara la camara
camare prepara(camara_nr,camara_nc,object.elemento(object.x,0,0,'L'),object elemento(object.y,0,0,'L')),
settextstyle(DEFAULT_FONT,HORIZ_DIR,1),                 //texto gráfico
outtextxy(140,20,"Periodo muestreo camara(s/100):");    //área de mensajes
outtextxy(140,40,"Periodo movimiento del objeto(s/100):"),
itoa(t_camara,camino1,10),
outtextxy(140+textwidth("Periodo muestreo camara(s/100).")+4,20,camino1);
itoa(t_objeto,camino1,10),
outtextxy(140+textwidth("Periodo movimiento del objeto(s/100):")+4,40,camino1),

//inicializa mouse
if(mouse.Exists()){
  mouse.xLimit(0,getmaxx());                    //fija limites del mouse
  mouse.yLimit(0,getmaxy());
  mouse.Enable();                               //habilita el mouse
  mouse.Move(3,136),                            //mueve el mouse a posición inicial
  mouse.MickToPix(64,128);                      //velocidad del mouse
  mouse.Show(),
}
else { exit(1); }

outtextxy(140,60,"Seleccione trayectoria...");          //mensaje
object.trayectoria(),                                   //genera la trayectoria
setviewport(140,60,140+textwidth("Seleccione trayectoria. ."),69,1),
clearviewport(),
setviewport(0,0,getmaxx(),getmaxy(),1),
object.pos_act_ren=camare.pos_act_ren=object.elemento(object y,0,0,'L'),  //establece posiciones iniciales de
object.pos_act_col=camare.pos_act_col=object.elemento(object.x,0,0,'L');  //objeto y camara según la trayectoria dada
object poner(object.elemento(object.y,0,0,'L'),object.elemento(object.x,0,0,'L')), //renglon, columna
A=imagen_nr;                    //renglones imagen
B=imagen_nc;                    //columnas imagen
indice=0,                       //inicio de la trayectoria
entrada=camare.captura(),       //captura la primera imagen
movedata(FP_SEG(entrada),FP_OFF(entrada),FP_SEG(auxiliar),FP_OFF(auxiliar),1L*imagen_nr*imagen_nc+10L),
auxiliar++;auxiliar++,auxiliar++;auxiliar++;            //apunta a donde empieza la imagen

//después de enfocar la escena a seguir, la camara coloca su centro de
//visión en el centroide del objeto

nbordes=0;
for(i=0;i<A,i++) {              //renglones imagen
  for(j=0;j<B;j++) {            //columnas imagen
        suma=suma_a=0;
        for(k=-(M-1)/2;k<=(M-1)/2,k++)          //renglones filtro
          for(l=-(M-1)/2;l<=(M-1)/2;l++) {      //columnas filtro
            y=i-k; x=j-l,                       //renglones, columnas
```

```
            if(y<0)                          //si no hay valor lo repite de la localidad simétrica
                y=abs(y);
            if(y>A-1)
                y=(A-1)-(y-i);
            if(x<0)                          //si no hay valor lo repite de la localidad simétrica
                x=abs(x);
            if(x>B-1)
                x=(B-1)-(x-j);
            pixelf=filtro[k+(M-1)/2][l+(M-1)/2];          //pixel del filtro
            pixeli=*(auxiliar+1L*y*B+x);                  //entrada[1L*y*B+x],
            suma=suma+(float)pixeli*pixelf;
          }
          if ((suma>=0 && suma_a>=0) || (suma<0 && suma_a<0)) {
            suma_a=suma;
          }
          else {
            *(coor+nbordes)=(unsigned char) j;            //coordenada x
            *(coor+nbordes+10000)=(unsigned char) i;      //coordenada y
            nbordes++;
          }
  }
}

//obtención del centroide
centroide_float[0][0]=centroide_float[0][1]=0;
for(m=0;m<nbordes;m++) {
  centroide_float[0][0]=centroide_float[0][0]+(float) (*(coor+m));          //coor[m][0];
  centroide_float[0][1]=centroide_float[0][1]+(float) (*(coor+10000+m));    //coor[m][1];
}
if(nbordes>0) {
  centroide_float[0][0]=centroide_float[0][0]/nbordes;
  centroide_float[0][1]=centroide_float[0][1]/nbordes;
}
centroide_int[0][0]=caux.x=(int)centroide_float[0][0];          //caux respalda el primer centroide
centroide_int[0][1]=caux.y=(int)centroide_float[0][1];          //en coordenadas relativas

//centroide inicial en coordenadas absolutas
c1.x=camare.pos_act_col+centroide_int[0][0];
c1.y=camare.pos_act_ren+centroide_int[0][1];

//ajuste del centro de visión de la camara con el centroide del objeto
camare.pos_act_col=camare.pos_act_col+(centroide_int[0][0]-camara_nc/2);
camare.pos_act_ren=camare pos_act_ren+(centroide_int[0][1]-camara_nr/2);
camare.pos_act_col=(camare.pos_act_col<=0)?1:camare.pos_act_col;
camare pos_act_ren=(camare.pos_act_ren<=133)?134:camare.pos_act_ren;
camara_x_aux=camare.pos_act_col;                    //respaldo de las posiciones iniciales
camara_y_aux=camare.pos_act_ren;                    //de la camara

//obtención de la siguiente imagen
entrada=camare.captura();
auxiliar=base;
movedata(FP_SEG(entrada),FP_OFF(entrada),FP_SEG(auxiliar),FP_OFF(auxiliar),1L*imagen_nr*imagen_nc+10L),
auxiliar++;auxiliar++;auxiliar++;auxiliar++;        //apunta a donde empieza la imagen
//termina ajuste de inicio

gettime(&tp),                                       //tiempo actual del sistema
                                                    //nuevo tiempo para mover objeto
tl_objeto=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+1L*t_objeto;
                                                    //nuevo tiempo para captura de la camara
tl_camara=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+t_camara,
do {
  mouse.Position();
  if(mouse.InBox(0,0,134,130)) {                    //sensado del menú con el mouse
        outtextxy(140,60,"Seleccione opcion con el mouse...");
        do {
          mouse.GetEvent();                         //pregunta si se ha presionado el boton izquierdo
          boton=mouse Pressed(0);                   //del mouse
```

```
        mouse.Position(),
      } while((boton==0) && (mouse InBox(0,0,134,130)));
      if(boton!=0) {
        mouse.Position();
        if(mouse.InBox(3,71,3+32,71+32)) opcion='T';
        else if(mouse.InBox(3,3,3+32,3+32)) opcion='O',
        else if(mouse InBox(3,37,3+32,37+32)) opcion='C',
        else if(mouse.InBox(45,3,45+32,3+32)) opcion='1';
        else if(mouse InBox(45,37,45+32,37+32)) opcion='2';
        else if(mouse.InBox(45,71,45+32,71+32)) opcion='3',
        else if(mouse.InBox(87,3,87+32,3+32)) opcion='4',
        else if(mouse.InBox(87,37,87+32,37+32)) opcion='S';
      }
      setviewport(140,60,140+textwidth("Seleccione opcion con el mouse..."),69,1),
      clearviewport();
      setviewport(0,0,getmaxx(),getmaxy(),1);
      mouse.Move(140,110),
      gettime(&tp),                                  //tiempo actual del sistema
                                                     //nuevo tiempo para mover objeto
      tl_objeto=1L*tp ti_hund+100L*tp.ti_sec+100L*60L*tp ti_min+100L*60L*60L*tp ti_hour+1L*t_objeto;
                                                    //nuevo tiempo para captura de la camara
      tl_camara=1L*tp.ti_hund+100L*tp ti_sec+100L*60L*tp ti_min+100L*60L*60L*tp ti_hour+t_camara,
}
if(opcion=='1' || opcion=='2' || opcion=='3' || opcion=='4' ) {          //selecciona nueva figura
      switch(opcion) {
        case '1'.strcpy(camino1,"fig1.gry"); break;
        case '2'.strcpy(camino1,"fig2.gry"), break;
        case '3':strcpy(camino1,"fig3.gry"); break;
        case '4':strcpy(camino1,"fig4.gry"), break,
      }
      object.lee(camino1,imagen_nr,imagen_nc),                   //lee la imagen de un archivo
       //prepara la camara
      camare prepara(camara_nr,camara_nc,object.elemento(object.x,0,0,'L'),object elemento(object.y,0,0,'L'));
      object pos_act_ren=camare.pos_act_ren=object elemento(object.y,0,0,'L'),   //inicializa posiciones iniciales
      object.pos_act_col=camare.pos_act_col=object.elemento(object.x,0,0,'L');   //de objeto y camara según la
                                                                                 //trayectoria dada
      setviewport(1,134,getmaxx()-1,getmaxy()-1,1);clearviewport();              //limpia el puerto de visión
      floodfill(3,136,1),                                  //rellena la zona de despliegue con color 255 negro
      setviewport(0,0,getmaxx(),getmaxy(),1);              //restablece puerto a pantalla completa
      indice=0;                                            //dibuja la trayectoria
      e1x=e2x=e1y=e2y=dex=dey=vel1x=vel1y=0;               //limpia variables del CLB
      object poner(object.elemento(object y,0,0,'L'),object elemento(object.x,0,0,'L'));     //renglon, columna
      entrada=camare.captura();                                                  //captura la primera imagen
      auxiliar=base;

      movedata(FP_SEG(entrada),FP_OFF(entrada),FP_SEG(auxiliar),FP_OFF(auxiliar),1L*imagen_nr*imagen_nc
               +10L),
      auxiliar++;auxiliar++,auxiliar++;auxiliar++,          //apunta a donde empieza la imagen

      //después de enfocar la escena a seguir, la camara coloca su centro de visión en el centroide del objeto
      nbordes=0;
      for(i=0;i<A;i++) {                                    //renglones imagen
        for(j=0;j<B,j++) {                                  //columnas imagen
          suma=suma_a=0;
          for(k=-(M-1)/2,k<=(M-1)/2;k++)                    //renglones filtro
                for(l=-(M-1)/2;l<=(M-1)/2;l++) {            //columnas filtro
                  y=i-k; x=j-l;                             //renglones, columnas
                  if(y<0)                                   //si no hay valor lo repite de la localidad simétrica
                   y=abs(y),
                  if(y>A-1)
                   y=(A-1)-(y-i),
                  if(x<0)                                   //si no hay valor lo repite de la localidad simétrica
                   x=abs(x),
                  if(x>B-1)
                   x=(B-1)-(x-j);
                  pixelf=filtro[k+(M-1)/2][l+(M-1)/2],      //pixel del filtro
                  pixeli=*(auxiliar+1L*y*B+x);              //entrada[1L*y*B+x],
```

```
                suma=suma+(float)pixeli*pixelf;
              }
          if ((suma>=0 && suma_a>=0) || (suma<0 && suma_a<0)) {
                suma_a=suma;
          }
          else {
                *(coor+nbordes)=(unsigned char) j;          //coordenada x
                *(coor+nbordes+10000)=(unsigned char) i; //coordenada y
                nbordes++;
          }
        }
      }

      //obtención del centroide
      centroide_float[0][0]=centroide_float[0][1]=0;
      for(m=0;m<nbordes;m++) {
        centroide_float[0][0]=centroide_float[0][0]+(float) (*(coor+m));          //coor[m][0];
        centroide_float[0][1]=centroide_float[0][1]+(float) (*(coor+10000+m));    //coor[m][1];
      }
      if(nbordes>0) {
        centroide_float[0][0]=centroide_float[0][0]/nbordes;
        centroide_float[0][1]=centroide_float[0][1]/nbordes;
      }
      centroide_int[0][0]=caux.x=(int)centroide_float[0][0];                      //caux respalda el primer centroide
      centroide_int[0][1]=caux.y=(int)centroide_float[0][1];                      //en coordenadas relativas
      //centroide inicial en coordenadas absolutas
      c1.x=camare.pos_act_col+centroide_int[0][0];
      c1.y=camare.pos_act_ren+centroide_int[0][1];

      //ajuste del centro de visión de la camara con el centroide del objeto
      camare.pos_act_col=camare.pos_act_col+(centroide_int[0][0]-camara_nc/2);
      camare.pos_act_ren=camare.pos_act_ren+(centroide_int[0][1]-camara_nr/2);
      camare.pos_act_col=(camare.pos_act_col<=0)?1:camare.pos_act_col;
      camare.pos_act_ren=(camare.pos_act_ren<=133)?134:camare.pos_act_ren,
      camara_x_aux=camare.pos_act_col;                                            //respaldo de las posiciones iniciales
      camara_y_aux=camare.pos_act_ren;                                            //de la camara
      //obtención de la siguiente imagen
      entrada=camare.captura();
      auxiliar=base;
      movedata(FP_SEG(entrada),FP_OFF(entrada),FP_SEG(auxiliar),FP_OFF(auxiliar),1L*imagen_nr*imagen_nc+10
            L);
      auxiliar++;auxiliar++;auxiliar++;auxiliar++;                                //apunta a donde empieza la imagen
      //termina ajuste de inicio

      gettime(&tp);                                                               //tiempo actual del sistema
                                                                                  //nuevo tiempo para mover objeto
      tl_objeto=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+1L*t_objeto;
                                                                                  //nuevo tiempo para capturar de la camara
      tl_camara=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+t_camara;
      opcion='z';                                                                 //no selección de ninguna opcion del menú
}
if(opcion=='t' || opcion=='T' ) {                                                 //selecciona nueva trayectoria
      outtextxy(140,60,"Seleccione trayectoria...");                              //mensaje
      setviewport(1,134,getmaxx()-1,getmaxy()-1,1);clearviewport();               //limpia el puerto de visión
      floodfill(3,136,1);                                                         //rellena con color 255 negro
      setviewport(0,0,getmaxx(),getmaxy(),1);                                     //restablece puerto a pantalla completa
       //prepara la camara
      camare.prepara(camara_nr,camara_nc,object.elemento(object.x,0,0,'L'),object.elemento(object.y,0,0,'L'));
      object.trayectoria();                                                       //genera la trayectoria
      object.pos_act_ren=camare.pos_act_ren=object.elemento(object.y,0,0,'L'); //establece posiciones iniciales
      object.pos_act_col=camare pos_act_col=object.elemento(object.x,0,0,'L');  //de objeto y camara según la
                                                                                  //trayectoria dada
      object.poner(object.elemento(object.y,0,0,'L'),object.elemento(object.x,0,0,'L')); //renglon, columna
      indice=0;                                                                   //inicio de la trayectoria
      e1x=e2x=e1y=e2y=dex=dey=vel1x=vel1y=0,                                      //limpia variables del CLB
      //ajuste del centro de visión de la camara con el centroide del objeto
      camare.pos_act_col=camare.pos_act_col+(caux.x-camara_nc/2);
```

```
        camare.pos_act_ren=camare.pos_act_ren+(caux.y-camara_nr/2),
        camare pos_act_col=(camare.pos_act_col<=0)?1.camare.pos_act_col,
        camare pos_act_ren=(camare.pos_act_ren<=133)?134:camare.pos_act_ren,
        camara_x_aux=camare.pos_act_col; //respaldo de las posiciones iniciales
        camara_y_aux=camare.pos_act_ren; //de la camara ya centrada
        c1.x=c2.x=64+camara_x_aux; c1.y=c2.y=64+camara_y_aux,          //recuperación del centroide inicial

        entrada=camare.captura();                                      //captura la primera imagen
        auxiliar=base;
        movedata(FP_SEG(entrada),FP_OFF(entrada),FP_SEG(auxiliar),FP_OFF(auxiliar),1L*imagen_nr*imagen_nc
                +10L);
        auxiliar++;auxiliar++;auxiliar++;auxiliar++,                    //apunta a donde empieza la imagen
        setviewport(140,60,140+textwidth("Seleccione trayectoria..."),69,1),
        clearviewport();
        setviewport(0,0,getmaxx(),getmaxy(),1);
        gettime(&tp);                          //tiempo actual del sistema
                                               //nuevo tiempo para mover objeto
        tl_objeto=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+1L*t_objeto,
                                               //nuevo tiempo para captura de la camara
        tl_camara=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+t_camara,
        opcion='z',
}
if(opcion=='o' || opcion=='O'|| opcion=='c' || opcion=='C') {    //selecciona nuevo periodo para el movimiento del objeto
        if(opcion=='o' || opcion=='O') {
          outtextxy(140,60,"Nuevo t_objeto (10..100).");
          outtextxy(140+textwidth("Nuevo t_objeto (10..100):")+4,60,"");
          int ind=0;
          char car;
          do {
            car=getch();
            if(car==13) {
                    camino1[ind]=NULL;
                    setviewport(140,60,140+textwidth("Nuevo t_objeto (10  100):")+4+80,60+9,1),
                    clearviewport();
                    setviewport(0,0,getmaxx(),getmaxy(),1),
            }
            else {
                    camino1[ind++]=car;
                    camino1[ind]=NULL;
                    setviewport(140+textwidth("Nuevo t_objeto (10..100).")+4,60,140+textwidth("Nuevo t_objeto
                            (10..100):")+4+80,60+9,1),
                    clearviewport(),
                    setviewport(0,0,getmaxx(),getmaxy(),1);
                    outtextxy(140+textwidth("Nuevo t_objeto (10. 100).")+4,60,camino1);
            }
          } while(car!=13),
          t_objeto=atoi(camino1);
          setviewport(140+textwidth("Periodo movimiento del objeto(s/100):")+4,40,140+textwidth("Periodo movimiento
                  del objeto(s/100):")+4+24,40+9,1),
          clearviewport();
          setviewport(0,0,getmaxx(),getmaxy(),1);
          outtextxy(140+textwidth("Periodo movimiento del objeto(s/100).")+4,40,camino1);
          gettime(&tp);                        //tiempo actual del sistema
                                               //nuevo tiempo para mover objeto
          tl_objeto=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+1L*t_objeto,
                                              //nuevo tiempo para captura de la camara
          tl_camara=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+t_camara;
          opcion='z',
        }
        if(opcion=='c' || opcion=='C') {
          outtextxy(140,60,"Nuevo t_camara (10..100):"),
          outtextxy(140+textwidth("Nuevo t_camara (10..100):")+4,60,""),
          int ind=0;
          char car;
          do {
            car=getch(),
            if(car==13) {
```

```
                camino1[ind]=NULL;
                setviewport(140,60,140+textwidth("Nuevo t_camara(10..100):")+4+80,60+9,1),
                clearviewport();
                setviewport(0,0,getmaxx(),getmaxy(),1);
            }
            else {
                camino1[ind++]=car;
                camino1[ind]=NULL;
                setviewport(140+textwidth("Nuevo t_camara(10..100):")+4,60,140+textwidth("Nuevo
                  t_camara(10..100):")+4+80,60+9,1);
                clearviewport();
                setviewport(0,0,getmaxx(),getmaxy(),1);
                outtextxy(140+textwidth("Nuevo t_camara(10..100):")+8,60,camino1);
            }
          } while(car!=13);
          t_camara=atoi(camino1);
          setviewport(140+textwidth("Periodo muestreo camara(s/100):")+4,20,140+textwidth("Periodo muestreo
                camara(s/100):")+4+24,20+9,1);
          clearviewport();
          setviewport(0,0,getmaxx(),getmaxy(),1);
          outtextxy(140+textwidth("Periodo muestreo camara(s/100):")+4,20,camino1);
          gettime(&tp);                                  //tiempo actual del sistema
                                                         //nuevo tiempo para mover objeto
          tl_objeto=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp ti_min+100L*60L*60L*tp.ti_hour+1L*t_objeto;
                                                         //nuevo tiempo para captura de la camara
          tl_camara=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+t_camara,
          opcion='z';
        }
}

//códigos de procesamiento
nbordes=0,
repite_t='N';                                    //aun no repite la trayectoria
switch(tec) {
        case 'L':                                //detección de bordes utilizando filtro Laplaciano
                for(i=0;i<A;i++) {               //renglones imagen
                  gettime(&tp);
                  //se verifica si hay que mover el objeto
                  if(1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour>tl_objeto) {
                        indice++;
                        if(object.elemento(object.y,indice,0,'L')==1023) {
                          indice=0;
                          c1.x=c2.x=64+camara_x_aux; c1.y=c2.y=64+camara_y_aux; //recuperación del centroide
                                                                                //inicial
                          setviewport(1,134,getmaxx()-1,getmaxy()-1,1);
                          clearviewport();
                          floodfill(3,136,1);                   //rellena zona de despliegue con 255, negro
                          setviewport(0,0,getmaxx(),getmaxy(),1);
                          object.poner(object.elemento(object.y,indice,0,'L'),object.elemento(object.x,indice,0,'L'));
                          //inicializa posiciones iniciales
                          object.pos_act_ren=camare.pos_act_ren=object.elemento(object.y,0,0,'L');
                          object.pos_act_col=camare.pos_act_col=object.elemento(object.x,0,0,'L');
                          camare.pos_act_ren=camara_y_aux;      //recuperación de las posiciones
                          camare.pos_act_col=camara_x_aux;      //de la camara
                            nbordes=0;
                          repite_t='S',                         //se ha comenzado a repetir la trayectoria
                          e1x=e2x=e1y=e2y=dex=dey=vel1x=vel1y=0;  //limpia variables del CLB
                          break;
                        }
                        else {
                          object.mover(object.elemento(object y,indice,0,'L'),object.elemento(object.x,indice,0,'L'),'N'),
                        }
                        tl_objeto=tl_objeto+t_objeto;
                  }
                  //se verifica si hay imagen que capturar
                  if(1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour>tl_camara) {
                        entrada=camare.captura();
```

```
                tl_camara=tl_camara+t_camara,
            }
            for(j=0;j<B;j++) {                              //columnas imagen
                suma=suma_a=0;
                for(k=-(M-1)/2;k<=(M-1)/2;k++)              //renglones filtro
                  for(l=-(M-1)/2;l<=(M-1)/2;l++) {          //columnas filtro
                    y=i-k; x=j-l,                           //renglones, columnas
                    if(y<0)                                 //si no hay valor lo repite de la localidad simétrica
                        y=abs(y);
                    if(y>A-1)
                        y=(A-1)-(y-i),
                    if(x<0)                                 //si no hay valor lo repite de la localidad simétrica
                        x=abs(x),
                    if(x>B-1)
                        x=(B-1)-(x-j);
                    pixelf=filtro[k+(M-1)/2][l+(M-1)/2],    //pixel del filtro
                    pixeli=*(auxiliar+1L*y*B+x);            //entrada[1L*y*B+x],
                    suma=suma+(float)pixeli*pixelf;
                  }
                if ((suma>=0 && suma_a>=0) || (suma<0 && suma_a<0)) {
                  suma_a=suma;
                }
                else {
                  *(coor+nbordes)=(unsigned char) j;         //coordenada x
                  *(coor+nbordes+10000)=(unsigned char) i,   //coordenada y
                  nbordes++;
                }
            }
          }
          break,
case 'F'  //Detección de bordes utilizando pseudo-lógica borrosa
          for(i=1,i<A-1;i++) {                  //renglones imagen
            gettime(&tp);
            //se verifica si se tiene que mover el objeto
            if(1L*tp ti_hund+100L*tp.ti_sec+100L*60L*tp ti_min+100L*60L*60L*tp.ti_hour>tl_objeto) {
                indice++,
                if(object.elemento(object.y,indice,0,'L')==1023) {
                  indice=0;
                  c1.x=c2 x=64+camara_x_aux; c1.y=c2 y=64+camara_y_aux, //recuperación del centroide
                                                                        //inicial
                  setviewport(1,134,getmaxx()-1,getmaxy()-1,1),
                  clearviewport(),
                  floodfill(3,136,1),                   //rellena zona de despliegue con 255, negro
                  setviewport(0,0,getmaxx(),getmaxy(),1),
                  object.poner(object elemento(object.y,indice,0,'L'),object.elemento(object x,indice,0,'L')),
                    //inicializa posiciones iniciales
                  object.pos_act_ren=camare.pos_act_ren=object.elemento(object.y,0,0,'L'),
                  object.pos_act_col=camare pos_act_col=object.elemento(object x,0,0,'L'),
                  camare.pos_act_ren=camara_y_aux;          //recuperación de las posiciones
                  camare.pos_act_col=camara_x_aux;          //de la camara
                  nbordes=0,
                  repite_t='S',                             //se ha comenzado a repetir la trayectoria
                  e1x=e2x=e1y=e2y=dex=dey=vel1x=vel1y=0;    //limpia variables del CLB
                  break,
                }
                else {
                  object.mover(object.elemento(object y,indice,0,'L'),object elemento(object.x,indice,0,'L'),'N'),
                }
                tl_objeto=tl_objeto+t_objeto;
            }
            //se verifica si hay que capturar imagen
            if(1L*tp ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour>tl_camara) {
                entrada=camare captura();
                tl_camara=tl_camara+t_camara;
            }
            for(j=1;j<B-1,j++) {                        //columnas imagen
                membresia[0]=OR_difusa(bajo(*(auxiliar+1L*i*B+(j-1))),medio(*(auxiliar+1L*i*B+(j-
```

```
                                        1))),alto(*(auxiliar+1L*i*B+(j-1))));          //(pixel x-1,y)
                membresia[1]=OR_difusa(bajo(*(auxiliar+1L*i*B+(j+1))),medio(*(auxiliar+1L*i*B+
                                        (j+1))),alto(*(auxiliar+1L*i*B+(j+1))));       //(pixel x+1,y)
              membresia[2]=OR_difusa(bajo(*(auxiliar+1L*(i-1)*B+j)),medio(*(auxiliar+1L*(i-
                                        1)*B+j)),alto(*(auxiliar+1L*(i-1)*B+j))));     //(pixel x,y-1)
                membresia[3]=OR_difusa(bajo(*(auxiliar+1L*(i+1)*B+j)),medio(*(auxiliar+1L*(i+1)*
                                        B+j)),alto(*(auxiliar+1L*(i+1)*B+j)));         //(pixel x,y+1)
              membresia[4]=OR_difusa(bajo(*(auxiliar+1L*(i)*B+j)),medio(*(auxiliar+1L*(i)*B+j)),alto(*(
                                        auxiliar+1L*(i)*B+j)));                        //(pixel x,y) pixel central
              if(!(membresia[4]==membresia[0] && membresia[4]==membresia[1] &&
                   membresia[4]==membresia[2] && membresia[4]==membresia[3]) ) {
                *(coor+nbordes)=(unsigned char) j;                                     //coordenada x
                *(coor+nbordes+10000)=(unsigned char) i;                               //coordenada y
                nbordes++;
              }
            }
          }
          break;
case 'C': //Detección de bordes mediante comparación
          for(i=1;i<A-1;i++) {                          //renglones imagen
            gettime(&tp);
            //se verifica si hay que mover el objeto
            if(1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour>tl_objeto) {
                indice++;
                if(object.elemento(object.y,indice,0,'L')==1023) {
                  indice=0;
                    //recuperación del centroide inicial
                  c1.x=c2.x=64+camara_x_aux; c1.y=c2.y=64+camara_y_aux;
                  setviewport(1,134,getmaxx()-1,getmaxy()-1,1);
                  clearviewport();
                  floodfill(3,136,1);                   //rellena zona de despliegue con 255, negro
                  setviewport(0,0,getmaxx(),getmaxy(),1);
                  object.poner(object.elemento(object.y,indice,0,'L'),object.elemento(object.x,indice,0,'L'));
                    //inicializa posiciones iniciales
                  object.pos_act_ren=camare.pos_act_ren=object.elemento(object.y,0,0,'L');
                  object.pos_act_col=camare.pos_act_col=object.elemento(object.x,0,0,'L');
                  camare.pos_act_ren=camara_y_aux;          //recuperación de las posiciones
                  camare.pos_act_col=camara_x_aux;          //de la camara
                    nbordes=0;
                  repite_t='S';                         //se ha comenzado a repetir la trayectoria
                  e1x=e2x=e1y=e2y=dex=dey=vel1x=vel1y=0; //limpia variables del CLB
                  break;
                }
                else {
                  object.mover(object.elemento(object.y,indice,0,'L'),object.elemento(object.x,indice,0,'L'),'N');
                }
                tl_objeto=tl_objeto+t_objeto;
            }
            //se verifica si hay que capturar imagen
            if(1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour>tl_camara) {
                entrada=camare.captura();
                tl_camara=tl_camara+t_camara;
            }
            for(j=1;j<B-1;j++) {                        //columnas imagen
                pixel_central=*(auxiliar+1L*i*B+j);     //pixel central
                membresia[0]=(unsigned char)abs((int)pixel_central-(int)(*(auxiliar+
                                1L*i*B+(j-1))));        //(pixel x-1,y)
                membresia[1]=(unsigned char)abs((int)pixel_central-(int)(*(auxiliar+1L
                                *i*B+(j+1))));          //(pixel x+1,y)
                membresia[2]=(unsigned char)abs((int)pixel_central-(int)(*(auxiliar+1L
                                *(i-1)*B+j)));          //(pixel x,y-1)
                membresia[3]=(unsigned char)abs((int)pixel_central-(int)(*(auxiliar+1L
                                 *(i+1)*B+j)));          //(pixel x,y+1)
                if(!(membresia[0]<=radio && membresia[1]<=radio && membresia[2]<=radio &&
                     membresia[3]<=radio)) {
                  *(coor+nbordes)=(unsigned char) j;    //coordenada x
                  *(coor+nbordes+10000)=(unsigned char) i;  //coordenada y
```

```
                                nbordes++;
                            }
                        }
                    }
                    break;
    }

    //obtención del centroide
    centroide_float[0][0]=centroide_float[0][1]=0;
    for(m=0,m<nbordes,m++) {
            centroide_float[0][0]=centroide_float[0][0]+(float) (*(coor+m)),          //coor[m][0],
            centroide_float[0][1]=centroide_float[0][1]+(float) (*(coor+10000+m)),    //coor[m][1];
    }
    if(nbordes>0) {
            centroide_float[0][0]=centroide_float[0][0]/nbordes,
            centroide_float[0][1]=centroide_float[0][1]/nbordes;
    }
    centroide_int[0][0]=(int)centroide_float[0][0],
    centroide_int[0][1]=(int)centroide_float[0][1],
    //centroide 2 en coordenadas absolutas
    if (nbordes!=0) {
            // predictor de velocidad constante
            c2.x=camare.pos_act_col+centroide_int[0][0];
            c2 y=camare.pos_act_ren+centroide_int[0][1];
            //predictor de velocidad con los centroides
            v1x=c2.x-c1.x;                                  //velocidad en dirección x
            v1y=c2 y-c1.y;                                  //velocidad en dirección y
            c1 x=c2.x;                                      //actualización de centroides
            c1 y=c2 y,
            // fin predictor de velocidad constante
            //Ajuste del centro de visión de la camara con el predictor borroso
            predictor_difuso(centroide_int[0][0],centroide_int[0][1],v1x,v1y,&inc_x,&inc_y);
            camare pos_act_col=camare.pos_act_col-inc_x+v1x,          //incluye el CLB
            camare pos_act_ren=camare.pos_act_ren-inc_y+v1y;
            camare.pos_act_col=(camare pos_act_col<=0)?1:camare pos_act_col,
            camare pos_act_ren=(camare pos_act_ren<=133)?134:camare.pos_act_ren,
    }
    // Si la imagen salió de cuadro entonces el centroide se fija en
    // [130,130] que significa objeto perdido
    if(nbordes==0)
            centroide_int[0][0]=centroide_int[0][1]=130;
    //escritura al archivo de datos
    if(repite_t=='N')
            fprintf(datos,"%d %d\n",centroide_int[0][0],centroide_int[0][1]);
    // Captura de nuevo cuadro
    auxiliar=base,                          //recuperación del apuntador
    entrada=camare.captura(),               //captura cuadro actual
    movedata(FP_SEG(entrada),FP_OFF(entrada),FP_SEG(auxiliar),FP_OFF(auxiliar),1L*imagen_nr*imagen_nc+10L),
    auxiliar++;auxiliar++;auxiliar++;auxiliar++;   //apunta a donde empieza la imagen
    //Si se comenzó de nuevo la trayectoria se inicializan tiempos
    if(nbordes==0) {
            gettime(&tp),                   //tiempo actual del sistema
                                            //nuevo tiempo para mover objeto
            tl_objeto=1L*tp.ti_hund+100L*tp.ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+1L*t_objeto;
                                            //nuevo tiempo para captura de la camara
            tl_camara=1L*tp.ti_hund+100L*tp ti_sec+100L*60L*tp.ti_min+100L*60L*60L*tp.ti_hour+t_camara;
    }
    // fin de Captura de nuevo cuadro
    if(kbhit()) opcion=getch();
  } while(!(opcion=='S' || opcion=='s'));
  farfree(coor);
  mouse Disable(),
  fclose(datos);
}

/*-------------------------Programa principal-------------------------*/
void main(void) {
```

```
  metodo tecnica;                                    //objeto con las técnicas a aplicar
  char camino[100]={"fig7.gry"};                     //archivo con la imagen por default
  clrscr();
  cout<<"Introduzca modo gráfico a inicializar: ";   //modo gráfico a manejar
  cout<<"\n 2->640*480*256";
  cout<<"\n 3->800*600*256";
  cout<<"\n 4->1024*768*256";
  cout<<"\nSELECCION: ";
  cin>>modo;
  modo=((modo!=2) && (modo!=3) && (modo!=4))?2:modo;
  tecnica.tecnica(camino,128,128,128,128,'L'),
  closegraph();
}
```

APÉNDICE B
DICCIONARIO DE DATOS Y CÓDIGO FUENTE DEL SOFTWARE PARA EL FUNCIONAMIENTO DEL SISTEMA FÍSICO GLOBAL.

DICCIONARIO DE DATOS.

Tabla B.1

Variable o Función	Tipo	Descripción
coor	Apuntador a entero no signado. Privado.	Almacena las coordenadas de los bordes [x,y].
entrada, I_bordes	Apuntador a caracter no signado. Privado.	Apuntador a la imagen adquirida y a la imagen de bordes.
tamx, tamy	Entero largo. Privado.	Dimensiones de la imagen.
MilApplication	MIL_ID. Privado.	Identificador de Aplicacion para el frame grabber.
MilSystem	MIL_ID. Privado.	Identificador de Sistema para el frame grabber.
MilDisplay	MIL_ID. Privado.	Identificador de despliegue para el frame grabber.
MilDigitizer	MIL_ID. Privado.	Identificador del Digitalizador para el frame

		grabber.
MilImage	MIL_ID. Privado.	Identificador del Buffer de almacenamiento para la Imagen.
buffer	MIL_ID. Privado.	Buffer de despliegue.
ventana	MIL_ID. Privado.	Identificador para la ventana de despliegue.
adquisitor(void)	---	Constructor
~adquisitor(void)	---	Destructor
obten_centroide(unsigned char radio, long *cx, long *cy)	Void	Obtiene el centroide de "imagen". Utilizando "radio" de comparación. Devuelve los resultado en "cx, cy".
despliega(void)	Void	Despliega el contenido de imagen de bordes "I_bordes".

Tabla B.1. Clase adquisitor.

Tabla B.1.1

Variable	Tipo	Descripción
nbordes	Entero no signado.	Número de puntos borde.
A,B	Enteros.	Dimensiones de la imagen de entrada. Renglones y columnas.
i,j	Enteros.	Contadores.
membresia	Arreglo de Enteros (4).	Membresía a la que pertenece cada pixel vecino del pixerl central.
pixel_central	Caracter no signado.	Valor del pixel central de la mascara.
m	Entero no signado.	Contador.
indice	Entero largo.	Indice para accesar las coordenadas de los bordes.

Tabla B.1.1. adquisitor.obten_centroide

Tabla B.2

Variable o Función	Tipo	Descripción
modo	Caracter no signado. Privado.	Modo de operacion de la electrónica.
pasos_x; pasos_y	Enteros no signados. Privado.	Número de pasos a efectuar en la coordenada X e Y.
control	Caracter no signado. Privado.	Byte de control para la electrónica.
contador	Entero largo. Privado.	Ajuste de ciclos por milisegundo.
puerto	Entero. Privado.	Dirección del puerto base.
prueba_x_grados, prueba_y_grados	Flotante. Privado.	Movimiento inicial en grados para el ajuste.
prueba_x_pixeles, prueba_y_pixeles	Flotante. Privado.	Movimiento inicial en pixels para el ajuste.
paso_x_en_grados, paso_y_en_grados	Flotante. Privado.	Equivalente de un paso en grados después del reductor de velocidad.
factor	Flotante. Privado.	Relacion del reductor de velocidad para ambas coordenadas.
electronica(void)	---	Constructor.
mueve_motores(unsigned char modo, unsigned int pasos_x,unsigned int pasos_y, unsigned char control)	Caracter no signado.	Ejecuta la rotación de los motores. Recibe el "modo" de operación de la electrónica, numero de pasos a rotar cada motor "pasos_x" y "pasos_y" y, la palabra de "control" que define el sentido y tamaño del paso.
retardo(long ms)	Void	Detiene la ejecución del programa "ms" milisegundos.
calibra(void)	Void	Efectua la calibracion necesaria para convertir la accion de control de pixeles a pasos de motor.
convierte_control_de_pixeles_a_pasos(int dux, int duy, float *pasosx, float	Void	Convierte la acción de control de pixeles ("dux" y "duy") a pasos motor

*pasosy)		("pasosx" y "pasosy").
posiciona_motores(float pasosx, float pasosy)	Void	Ejecuta que los motores roten "pasosx" y "pasosy".

Tabla B.2. Clase electronica.

Tabla B.2.1

Variable	Tipo	Descripción
timer, limite	Enteros largos.	Contadores para controlar el retardo.

Tabla B.2.1. electronica.retardo

Tabla B.2.2

Variable	Tipo	Descripción
x_inicio, y_inicio, x_fin, y_fin	Enteros largos.	Centroides del objeto antes y después de efectuar el movimiento de calibración.
pasosx, pasosy	Flotantes.	Numero de pasos necesarios para centrar la primera imagen utilizada para la calibración.

Tabla B.2.2. objeto.calibra

Tabla B.2.3

Variable	Tipo	Descripción
dux_grados, duy_grados	float	Acción de control en grados.

Tabla B.2.3. electronica.convierte_control_de_pixeles_a_pasos

Tabla B.2.4

Variable	Tipo	Descripción
control	Caracter no signado.	Palabra de control para la electrónica de control de motores.
no_pasos_x, no_pasos_y	Entero no signado.	Número de pasos a ejecutar por coordenada.

Tabla B.2.4. electronica.posiciona_motores

Tabla B.3

Variable o Función	Tipo	Descripción
e1x, e2x	Floatante. Privada.	Error actual (2) y anterior (1) en la coordenada X (columnas) como entrada para el CLB.
e1y, e2y	Floatante. Privada.	Error actual (2) y anterior (1) en la coordenada Y (renglones) como entrada para el CLB.
dex, dey	Floatante. Privada.	Variación del error para la coordenada X e Y como entrada para el CLB.
v1x, v1y	Floatante. Privada..	Variables de velocidad en la coordenada X e Y como entrada para el CLB.
pos_act_x, pos_act_y, pos_ant_x, pos_ant_y	Enteras. Privadas	Centroides anterior y actual de la imagen en coordenadas relativas al campo de visión de la cámara.
posx_logica_camara, posy_logica_camara	Enteras. Privadas	Posicion lógica de la camara en pixeles.
xs	Flotante. Privada.	Parámetro para la variacion de la simetria de las funciones de membresia en el intervalo [0,1]
alfa, beta	Flotante. Privadas.	Parámetros para la función de membresía exponencial.
corrimiento	Flotante. Privada.	Parámetro de corrimiento para la función de membresía exponencial.
velocidadx, aceleracionx, velocidady, aceleraciony	Flotante. Privadas.	Variables de entrada "velocidad" y "aceleración" para el CLB utilizado para controlar la ganancia.
pos_act_x, pos_act_y, pos_ant_x, pos_ant_y	Enteras.	Centroides de las imágenes anterior y actual en coordenadas absolutas.
Fuzzy_Logic_Controller(void)	---	Constructor.
negativo(float x)	Flotante.	Funcion de membresía "Negativo".

zero(float x)	Flotante.	Función de membresía "Zero".
positivo(float x)	Flotante.	Función de membresía "Positivo".
min_and(float a, float b, float c)	Flotante.	Operador And borroso. Selecciona el mínimo de sus argumentos.
positivo_vel_e(float x)	Flotante.	Función de membresía exponencial "Positivo" para la velocidad del objeto.
negativo_vel_e(float x)	Flotante.	Función de membresía exponencial "Negativo" para la velocidad del objeto.
ganancia_dinamica_FLC(int *gananciax, int *gananciay)	Void	Devuelve en "gananciax" y "gananciay" una ganancia dinámica calculada mediante un CLB.
Accion_de_Control(int xo, int yo, int *dux, int *duy)	Void	Recibe el centroide del objeto en "xo" y "yo". Proporciona la accion de control a aplicar a la posicion de la cámara en los apuntadores "dux" y "duy" para las coordenadas X e Y respectivamente.

Tabla B.3. Clase Fuzzy_Logic_Controller.

Tabla B.3.1

Variable	Tipo	Descripción
kv	Flotante.	Constante de normalización para la variable de velocidad.
kdv	Flotante.	Constante de normalización para la variable derivada de la velocidad (aceleración).
u	Arreglo Flotante. (9).	DOF (degree of firing), son nueve reglas.
fi	Arreglo bidimensional flotante (9)(3).	Resultados de la evaluación de cada regla. Funciones de salida singleton, son nueve reglas.
F	Arreglo Flotante. (3).	Función de salida global. Contiene el resultado de la agregación de reglas.
i	Entero.	Contador.

Tabla B.3.1. metodo Fuzzy_Logic_Controller.ganancia_dinamica_FLC

Tabla B.3.2

Variable	Tipo	Descripción
wx, wy	Flotantes.	Referencias para el control en las coordenadas X e Y.
ke	Flotantes.	Constante de normalizacion para la variable de entrada "error" al CLB.
kd	Flotantes.	Constante de normalizacion para la variable de entrada "derivada del error" al CLB.
kdu	Flotantes.	Constante de normalizacion para la variable de salida del CLB.
kv	Flotantes.	Constante de normalizacion para la variable de entrada "velocidad" al CLB.
vel1x, vel1y	Flotantes.	Velocidades normalizadas para la velocidad del objeto en las coordenadas X e Y.
u	Arreglo flotante (9).	DOF (degree of firing), son nueve reglas
fi	Arreglo bidimensional flotante (9)(3).	Resultados de la evaluación de cada regla. Funciones de salida singleton, son nueve reglas.
F	Arreglo flotante (3).	Función de salida global. Contiene el resultado de la agregación de reglas.
i	Entero.	Contador
ganx, gany	Enteros.	Registros de las ganancias a aplicar a cada una de las coordenadas por el CLB.

Tabla B.3.2. metodo Fuzzy_Logic_Controller.ganancia_dinamica_FLC.Accion_de_Control.

CÓDIGO FUENTE.

```
/*****************************************************************************************************/
/*----------------------Autor. Jorge Valeriano Assem-------------------------------------------------*/
/*------Software para el funcionamiento del Sistema Físico Global. (Sistema de Vision)---*/

/* NOTAS. * Obtencion de bordes por comparacion de minima distancia.
          * Comandos de control para la electronica de control de movimiento de la
            camara de video en el pto. 4F0H
FIN NOTAS*/

/*---------------------Inclusion de archivos headers----------------------*/

#include<stdio h>
#include<iostream h>
#include<stdlib.h>
#include<conio.h>
#include<except h>
#include<math.h>
```

```
#include<dos.h>
#include <mil.h>   //Libreria para el manejo del frame grabber.


/*-------------------Declaración de objetos-----------------------------*/

// Esta clase adquiere, procesa y calcula el centroide de la imagen.
class adquisitor {
   unsigned int huge *coor;                          //coordenadas de los bordes [x,y], [30000][2]
   unsigned char huge *entrada,huge *l_bordes,       //apuntador a la imagen adquirida y a la imagen de bordes
   long tamx, tamy;                                  //dimensiones de la imagen
   MIL_ID  MilApplication,                           /* Identificador de Aplicacion*/
   MilSystem,                                        /* Identificador de Sistema*/
   MilDisplay,                                       /* Identificador de despliegue*/
   MilDigitizer,                                     /* Identificador del Digitalizador*/
   MilImage;                                         /* Identificador para el Bufer de la Imagen*/
   MIL_ID  buffer;                                   /* Buffer de despliegue*/
   MIL_ID  ventana;                                  /* Identificador para la ventana de despliegue */
 public:
   adquisitor(void);                                 //constructor
   void obten_centroide(unsigned char radio, long *cx, long *cy);      //obtiene el centroide
   void despliega(void);                             //despliega la imagen adquirida
   ~adquisitor(void);                                //destructor
};


//Esta clase envia informacion a la electronica para el movimiento de la camara
class electronica {
   unsigned char modo,                                  //modo de operacion de la electronica
   unsigned int pasos_x;                                //numero de pasos a dar en la coordenada X
   unsigned int pasos_y;                                //numero de pasos a dar en la coordenada Y
   unsigned char control,                               //byte de control para la electronica
   long contador;                                       //ajuste de ciclos por milisegundo
   int puerto;                                          //puerto base
   float prueba_x_grados, prueba_y_grados;              //movimiento inicial en grados para el ajuste
   float prueba_x_pixeles, prueba_y_pixeles;            //movimiento inicial en pixels para el ajuste
   float paso_x_en_grados, paso_y_en_grados,            //equivalente de un paso en grados en la flecha del motor
   float factor;          //relacion entre flecha del motor y eje de la camara el mismo para ambas coordenadas
 public.
   electronica(void);                                   //constructor
   unsigned char mueve_motores(unsigned char modo, unsigned int pasos_x,unsigned int pasos_y, unsigned char
                               control);                //desplaza los motores un número de pasos
   void retardo(long ms);                               //retardo en milisegundos.
   void calibra(void);                  //Hace la calibracion para convertir la accion de control de pixeles a pasos
   void convierte_control_de_pixeles_a_pasos(int dux, int duy, float *pasosx, float *pasosy); //convierte el control de
                                                                                              //pixeles a pasos completos
   void posiciona_motores(float pasosx, float pasosy); //Efectua el movimiento fisico de los motores
};


//Esta clase se encarga del control borroso para la posicion de la camara
class Fuzzy_Logic_Controller {
   float e1x,e2x;                  //error actual(2) y anterior(1) en x (columnas) para el FLC
   float e1y,e2y;                  //error actual(2) y anterior(1) en y (renglones) para el FLC
   float dex,dey;                  //derivadas (variaciones) del error  para el FLC
   float v1x,v1y;                  //velocidad para el FLC
   int pos_act_x,pos_act_y,pos_ant_x,pos_ant_y ;     //centroides anterior y actual en coordenadas absolutas
   int posx_logica_camara,posy_logica_camara;        //posicion logica de la camara en pixeles
   float xs;                       //variacion de la simetria de las funciones de membresia (0,1)
   float alfa,beta;                //parametro para la exponencial
   float corrimiento;              //parametro para la exponencial
   float beta_ganancia;            //beta para la ganancia dinamica
   float velocidadx,aceleracionx, velocidady, aceleraciony;     //datos para el FLC corrector de ganancia
 public:
   int pos_act_x,pos_act_y,pos_ant_x,pos_ant_y ;  //centroides anterior y actual en coordenadas absolutas
   Fuzzy_Logic_Controller(void),                  //constructor
   float negativo(float x);                       //funcion de membresia
```

```
    float zero(float x);                                    //funcion de membresia
    float positivo(float x),                                //funcion de membresia
    float min_and(float a, float b, float c),               //and difusa, selecciona el minimo
    float positivo_vel_e(float x),                          //funcion de membresia para la velocidad
    float negativo_vel_e(float x),                          //funcion de membresia para la velocidad
    void ganancia_dinamica_FLC(int *gananciax, int *gananciay),  //devuelve una ganancia dinamica calculada con FLC
    void Accion_de_Control(int xo, int yo, int *dux, int *duy);        //Proporciona la accion de control
};


/*-------------------------Objetos globales-----------------------------------*/

adquisitor imagen;                                  // objeto para la adquisicion y procesamiento de la imagen
electronica puerto_4f0h,                            //objeto para control de la electronica de motores
Fuzzy_Logic_Controller controlador;                  //objeto para el controlador

/*-----------------------Definicion de funciones-----------------------------*/

/*Este constructor inicializa las variables dinamicas y el digitalizador */

adquisitor::adquisitor(void) {
  tamx=512; tamy=480,                          //tamaño de la imagen en pixeles
  /*Aloja el sistema de digitalizacion*/
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem,M_NULL, &MilDigitizer, &MilImage),
  /* Aloja el buffer de despliegue */
  if(MbufAlloc2d(M_DEFAULT,(tamx-6)/3,(tamy-6)/3,8+ M_UNSIGNED, M_IMAGE+ M_DISP+ M_GRAB, &buffer)
    ==NULL) {
    printf("Error al alojar el buffer...");
    exit(-1);
  }
 /* Aloja el display */
 if(MdispAlloc(MilSystem,M_DEFAULT,M_DISPLAY_SETUP,M_DEFAULT,&ventana)==M_NULL) {
    printf("Error al alojar el display.. ");
    exit(-1),
  }
  try {
    entrada=new unsigned char huge [1L*tamx*tamy+40L];
    l_bordes=new unsigned char huge [(tamx-6)*(tamy-6)/9+40L],
    coor=new unsigned int huge[30000L*2L],
  }
  catch (xalloc) {
    cout << "No hay memoria suficiente (1).. ";
    exit(-1);
  }
  printf("\npase adquisitor presiona enter");
  while(!kbhit());
  getc(stdin);
}


/*Este destructor libera las variables dinamicas*/

adquisitor::~adquisitor(void) {
  delete entrada;
  delete l_bordes,
  delete coor,
  /* Libera el digitalizador */
  MbufFree(buffer);
  MdispFree(ventana);
  MappFreeDefault(MilApplication, MilSystem, M_NULL,MilDigitizer, MilImage);
}


/*Esta funcion despliega la imagen de bordes procesadas */

void adquisitor::despliega(void) {
  /*Pone Imagen en el Bufer de Despliegue y despliega la imagen */
```

```
  MbufPut(buffer,I_bordes);
  MdispSelect(ventana,buffer);
}

/*Este metodo adquiere la imagen, la procesa y entrega centroide de bordes en las variables
cx y cy (columnas y renglones), recibe el radio de comparacion a usar */

void adquisitor::obten_centroide(unsigned char radio, long *cx, long *cy) {
  unsigned int nbordes;                    //numero de puntos borde
  int A,B;                                 //dimensiones de la imagen de entrada. Renglones y columnas.
  int i,j;                                 //contadores.
  int membresia[4];                        //membresia a la que pertenece
  unsigned char pixel_central;             //valor del pixel central de la mascara
  unsigned int m;                          //contador
  long indice;                             //indices para el arreglo

  /* Graba una imagen */
  MdigGrab(MilDigitizer, MilImage);
  /* Pasa la imagen grabada a un arreglo */
  MbufGet(MilImage,entrada);
  A=(int)tamy; B=(int)tamx;
  nbordes=0;
  for(i=4;i<A-4;i=i+3) {              //renglones imagen
    for(j=4;j<B-4;j=j+3) {            //columnas imagen
      pixel_central=*(entrada+1L*i*B+j);//imagen[1L*i*B];          //pixel central
      membresia[0]=abs(pixel_central-(*(entrada+1L*i*B+(j-3))));  //(pixel x-1,y)
      membresia[1]=abs(pixel_central-(*(entrada+1L*i*B+(j+3))));  //(pixel x+1,y)
      membresia[2]=abs(pixel_central-(*(entrada+1L*(i-3)*B+j)));  //(pixel x,y-1)
      membresia[3]=abs(pixel_central-(*(entrada+1L*(i+3)*B+j)));  //(pixel x,y+1)
      indice=1L*((B-6)/3)*(i-4)/3+(j-4)/3;
      if(membresia[0]<=radio && membresia[1]<=radio && membresia[2]<=radio && membresia[3]<=radio) {
        *(I_bordes+indice)=0,
      }
      else {
        *(coor+nbordes)=j;                //coordenada x
        *(coor+nbordes+30000)=i;          //coordenada y
        nbordes++;
        *(I_bordes+indice)=255;
      }
    }
  }
  *cx=*cy=0;
  for(m=0;m<nbordes;m++) {
    *cx=*cx + (*(coor+m));                //coor[m][0];
    *cy=*cy + (*(coor+30000+m));          //coor[m][1];
  }
  if(nbordes>0) {
    *cx=*cx/nbordes;
    *cy=*cy/nbordes;
  }
  else {
    *cx=85; *cy=80 ;
  }
}


/* Este constructor inicializa la electronica */

electronica::electronica(void) {
  puerto=0x4f0;                       //direccion base del puerto
  struct time s1,s2;
  long centesimas;
  long limite=1000000L;               //limite del ciclo de ajuste del retardo

  gettime(&s1);
  for(contador=0;contador<limite;contador++);
  gettime(&s2);
```

```
  centesimas=(s2.ti_hund+s2.ti_sec*100)-(s1.ti_hund+s1 ti_sec*100);
  centesimas=centesimas*10L;          //ahora esta en mseg
  contador= limite/centesimas;        //contador por milisegundo aprox. 3030 cuentas por mseg.
  mueve_motores(3,0,0,0);             //inicializa 8255A
  retardo(3L);
  printf("\nInicialice elec, conecta cable y pulsa tecla y muevo mot.");
  while(!kbhit());
  getc(stdin),
  mueve_motores(1,200,200,0*1 + 0*2 + 1*4 + 1*8);   //mueve 200 pasos horario
  printf("\nesperando termine movimiento");
  while(mueve_motores(2,0,0,0)==0),
  retardo(4000L); /*bajarlo a 1000*/
  printf("\nesperando terminen movimiento");
  mueve_motores(1,200,200,0*1 + 0*2 + 0*4 + 0*8);   //mueve 200 pasos antihorario
  while(mueve_motores(2,0,0,0)==0);
  // Inicializa variables para la calibracion de pixel->paso
  prueba_x_grados= prueba_y_grados=45.0,        //45 grados para el ajuste en eje de la camara
  prueba_x_pixeles= prueba_y_pixeles=0 0;
  paso_x_en_grados= paso_y_en_grados=1 8;    //Un paso completo equivale a 1.8 grados en el eje del motor
  factor=1 0/5.0; //relacion entre flecha motor y eje camara (5 a 1)
}
```

/* Este funcion se encarga del control de los motores para dar el movimiento a la estructura mecanica. Se utiliza la interface periferica 8255A que consta de tres puertos paralelos, A B C, los puertos A, B son
de salida y el C low es de entrada, C hi de salida, todos los puertos son de ocho bits.
La funcion puede hacer dos labores:
1) Si la variable Modo=0, se resetea el circuito.
2) Si modo=1, Desplaza los motores independientemente por coordenada, el numero de pasos indicados en pasos_x y pasos_y Haciendo uso de los parametros. x_paso (1 medio paso, 0 paso completo), y_paso (1 medio paso, 0 paso completo), x_sentido (1 horario es decir derecha, 0 antihorario es decir izquierda), y_sentido (1 horario es decir derecha, 0 antihorario es decir izquierda). Estos parametros se pasan en la variable control, en los 4 bits menos significativos respectivamente.
3) Si modo=2, la funcion devuelve un 1 si la electronica ya termino de efectuar el control anterior, 0 si aun no termina.
4) Si modo=3 entonces se inicializa la interface y la electronica

0	1	2	3
x_paso	y_paso	x_sentido	y_sentido
1=medio	1=medio	1=horario	1=horario
0=completo	0=completo	0=antihor	0=antihor

direccion base 04f0 */

```
unsigned char electronica::mueve_motores(unsigned char modo, unsigned int pasos_x,unsigned int pasos_y, unsigned char control) {
  switch(modo) {
    case 3. //inicializa la interface y la electronica
            outportb(puerto+3,0x81);          //programa a ptos. A B C Hi como salida C Low como entrada
            retardo(3L);                      //retardo en mseg
            outportb(puerto+2,0xf0);          //Para el astable en x e y
            retardo(3L),
            break;
    case 0: //resetea la electronica
            outportb(puerto+2,0xf0);          //para el astable en x e y
            retardo(3L);
            break,
    case 2: //checa si la electronica completo el movimiento
            unsigned char a;
            a=inportb(puerto+2);
            a=a&0x01;                         //extrae el primer bit
            return(a);
    case 1: //baja la informacion a los motores e inicia el movimiento
            unsigned char b,
            //deshabilita el start x e y
            retardo(3L);
            outportb(puerto+2,0x30),
            retardo(3L);
            // carga parte baja de coordenada X
            b=0x00ff & pasos_x;outportb(puerto+1,0xe0); outportb(puerto+0,b);
```

```
            retardo(3L);
            outportb(puerto+1,0xf0);
            retardo(3L),
            // carga parte alta de la coordenada X
            b=(0xff00 & pasos_x) >> 8;outportb(puerto+1,0xd0); outportb(puerto+0,b);
            retardo(3L);
            outportb(puerto+1,0xf0);
            retardo(3L);
            // carga parte baja de coordenada Y
            b=0x00ff & pasos_y;outportb(puerto+1,0xb0), outportb(puerto+0,b);
            retardo(3L);
            outportb(puerto+1,0xf0);
            retardo(3L);
            // carga parte alta de la coordenada Y
            b=(0xff00 & pasos_y) >> 8;outportb(puerto+1,0x70); outportb(puerto+0,b);
            retardo(3L);
            outportb(puerto+1,0xf0);
            retardo(3L),
            // establece sentidos y tamano de los pasos
            b=control; b=0xf0 | control; outportb(puerto+1,b);
            retardo(3L);
            // Inicia el movimiento
            if ((pasos_x!=0) && (pasos_y==0)) {
              outportb(puerto+2,0x70),                  // el astable en X
              retardo(3L);                              //retardo para detener el astable
            }
            if ((pasos_x==0) && (pasos_y!=0)) {
              outportb(puerto+2,0xb0);                  //detiene el astable en Y
              retardo(3L);                              //retardo para detener el astable
            }
            if ((pasos_x!=0) && (pasos_y!=0)) {
              outportb(puerto+2,0xf0),                  // detiene el astable en X e Y
              retardo(3L);                              //retardo para detener el astable
            }
            if ((pasos_x==0) && (pasos_y==0)) {
              outportb(puerto+2,0x30);                  // no detiene los astables en las coordenadas X e Y
              retardo(3L);
            }
            retardo(3L);
            outportb(puerto+2,0x30);                    //deja listo el momoestable para disparo
            retardo(3L);
            //outportb(0x302,0xf0);
            // empieza
            if ((pasos_x!=0) && (pasos_y==0))
              outportb(puerto+2,0x20);                  //empieza en X
            if ((pasos_x==0) && (pasos_y!=0))
              outportb(puerto+2,0x10);                  //empieza en Y
            if ((pasos_x!=0) && (pasos_y!=0))
              outportb(puerto+2,0x00);                  //empieza en X e Y
            if ((pasos_x==0) && (pasos_y==0))
              outportb(puerto+2,0x30);                  //no empieza empieza X ni Y
          break;
  }
  return(0);
}

/* Esta función genera un retardo en milisegundos*/

void electronica::retardo(long ms) {
  long timer, limite,
  limite=ms*contador;
  for(timer=0;timer<limite;timer++);
}


// Esta funcion calibra parametros para hacer una traduccion de pixeles a pasos
// ademas centra la primera imagen
```

```
void electronica::calibra(void) {
  long x_inicio, y_inicio, x_fin, y_fin,
  float pasosx,pasosy,                          //pasos para el centrado de la primera imagen
  retardo(4000L);
  imagen.obten_centroide(30,&x_inicio,&y_inicio);
  printf("\npase calculo del centroide, espero termine movimiento 45"),
  // mueve prueba_x_grados y prueba_y_grados en el eje de la camara
  mueve_motores(1,(int)((1.0/factor)*(prueba_x_grados)/(paso_x_en_grados)), (int) ((1 0/factor)*(prueba_y_grados)
                  /(paso_y_en_grados)), 0*1 + 0*2 + 1*4 + 1*8);
  while(mueve_motores(2,0,0,0)==0),
  imagen.obten_centroide(30,&x_fin,&y_fin),
  prueba_x_pixeles=(float)labs(x_inicio-x_fin);
  prueba_y_pixeles=(float)labs(y_inicio-y_fin);
  retardo(4000L);
  printf("\npase calculo 2 de centroide y espero se muevan motores");
  //centra la imagen
  convierte_control_de_pixeles_a_pasos((int)(x_fin)-(512/2),(int)(y_fin)-(480/2),&pasosx,&pasosy),
  posiciona_motores(pasosx,pasosy),
  controlador.pos_act_x=512/2; controlador.pos_act_y=480/2;      //establece posiciones actuales
  while(mueve_motores(2,0,0,0)==0),
}


//Esta funcion recibe las acciones de control en pixeles y las convierte a pasos completos

void electronica :convierte_control_de_pixeles_a_pasos(int dux, int duy, float *pasosx, float *pasosy) {
  float dux_grados, duy_grados,
  dux_grados=((float)dux)*prueba_x_grados/prueba_x_pixeles;
  duy_grados=((float)duy)*prueba_y_grados/prueba_y_pixeles;
  *pasosx=(1 0/factor)*dux_grados/paso_x_en_grados;          //control en pasos completos  x
  *pasosy=(1 0/factor)*duy_grados/paso_y_en_grados;          //control en pasos completos  y
}

//Esta funcion efectua el movimiento real en los motores para la accion de control ya
//recibida en pasos, debe calcular el sentido con base en el signo de la acción de control, asi
//como si utilizar medios pasos o pasos completos según sea el caso
// modo=1,
//                    0          1          2          3
//                  x_paso     y_paso    x_sentido  y_sentido
//                  1=medio    1=medio   1=horario  1=horario
//                  0=completo 0=completo 0=antihor. 0=antihor

void electronica::posiciona_motores(float pasosx, float pasosy) {
  unsigned char control=0;                  //palabra de control para la electronica
  unsigned int no_pasos_x,no_pasos_y,       //pasos en valor entero a aplicar
  if(fabs(pasosx)>50 0) {                   //pasos completos  X
    control=control&0x0e;
    no_pasos_x=(unsigned int)pasosx,
  }
  else {                                    //medios pasos   X
    control=control|0x01;
    no_pasos_x=(unsigned int)(pasosx*2.0);
  }
  if(fabs(pasosy)>50.0) {                   //pasos completos  Y
    control=control&0x0d,
    no_pasos_y=(unsigned int)pasosy,
  }
  else {                                    //medios pasos   Y
    control=control|0x02;
    no_pasos_y=(unsigned int)(pasosy*2 0),
  }
  if(pasosx<0 0)                            // X antiorario
    control=control&0x0b ;
  else
    control=control|0x04,                   // X horario
  if(pasosy<0 0)                            // Y horario  ( en una imagen Y crece hacia abajo )
```

```
    control=control|0x08;
  else
    control=control&0x07 ;                    // Y antihorario
    mueve_motores(1, no_pasos_x, no_pasos_y, control);
}


/* Constructor para las variables del Fuzzy Logic Controller */

Fuzzy_Logic_Controller::Fuzzy_Logic_Controller(void) {
  e1x=e2x=e1y=e2y=dex=dey=v1x=v1y=0;
  velocidadx=aceleracionx=velocidady=aceleraciony=0;
  pos_act_x=pos_act_y=pos_ant_x=pos_ant_y=0;
  posx_logica_camara=posy_logica_camara=0;
}

 /*Funcion de mebresia para el FLC */

float Fuzzy_Logic_Controller::negativo(float x) {
  if(x<=-xs)
    return(1.0);
  else
  if((-xs<x) && (x<=0))
    return((-1.0/xs)*x);
  return(0);
}

/* Funcion de mebresia para el FLC   */
float Fuzzy_Logic_Controller::zero(float x) {
  if((-xs<=x) && (x<0))
    return((1.0/xs)*x+1);
  else
    if((0<=x) && (x<=xs))
      return((-1.0/xs)*x+1);
  return(0);
}

/* Función de mebresia para el FLC */

float Fuzzy_Logic_Controller.:positivo(float x) {
  if((0<=x) && (x<xs))
    return((1.0/xs)*x);
  else
  if(x>=xs)
    return(1.0);
  return(0),
}

/* Funcion de membresia para el FLC */
float Fuzzy_Logic_Controller.:positivo_vel_e(float x) {          //funcion de membresia para la velocidad
  if((-1.0+corrimiento<=x) && (x<=corrimiento))
    return((float)(alfa*exp(beta*(x-corrimiento))));
  else
  if(x>corrimiento)
    return(1 0);
  return(0);
}

/* Funcion de membresia para el FLC */

float Fuzzy_Logic_Controller::negativo_vel_e(float x) {          //funcion de membresia para la velocidad
  if((-corrimiento<=x) && (x<=1.0-corrimiento))
    return((float)(alfa*exp(-beta*(x+corrimiento))));
  else
  if(x<-corrimiento)
    return(1.0);
  return(0);
```

```
}

/* Esta funcion efectua la and difusa para el FLC, selecciona el minimo */

float Fuzzy_Logic_Controller::min_and(float a, float b, float c) {
  if((a<=b) && (a<=c))
    return(a),
  else
  if((b<=a) && (b<=c))
    return(b);
  else
    return(c);
}

//Devuelve una ganancia dinamica calculada con un FLC, tres funciones de
//membresia y nueve reglas.

void Fuzzy_Logic_Controller::ganancia_dinamica_FLC(int *gananciax, int *gananciay) {
  float kv=1.0/94.0;                    //constante de normalizacion para la velocidad
  float kdv=1.0/60.0;                   //constante de normalizacion para la derivada de la velocidad, aceleracion
  float u[9];                           //DOF (degree of firing), son nueve reglas
  float fi[9][3]={{0,0,0},
                  {0,0,0},
                  {0,0,0},
                  {0,0,0},
                  {0,0,0},
                  {0,0,0},
                  {0,0,0},
                  {0,0,0},
                  {0,0,0}};             //funciones de salida por regla,(singleton), son nueve reglas
  float F[3]={0,0,0};                   //funcion de salida global
  int i;                                //contador
  // parametro de simetria de las funciones de membresia
  xs=0.5;
  //Actualizaciones
  aceleracionx=v1x-1.0*velocidadx/kv, aceleracion y=v1y-1.0*velocidady/kv,
  aceleracionx=kdv*aceleracionx; aceleracion y=kdv*aceleracion y;
  velocidadx=kv*v1x; velocidady=kv*v1y;
  /*gotoxy(20,11); printf("                                                    ");
  gotoxy(20,11), printf("vel x %f  en y %f",velocidadx,velocidady),
  gotoxy(20,12); printf("                                                    ");
  gotoxy(20,12); printf("acel x %f  en y %f",aceleracionx,aceleracion y);*/

  //EVALUACION DE LAS REGLAS PARA X
  //Regla 0 Si v(k) es negativo y dv(k) es negativo y kdu es negativo
  u[0]=min_and(negativo(velocidadx),negativo(aceleracionx),1.0),   //DOF
  fi[0][2]=u[0];                                                    //u^d, corte de la funcion de salida
  //Regla 1 Si v(k) es cero y dv(k) es negativo entonces kdu es negativo
  u[1]=min_and(zero(velocidadx),negativo(aceleracionx),1 0);        //DOF
  fi[1][0]=u[1];                                                    //u^d, corte de la funcion de salida
  //Regla 2 Si v(k) es positivo y dv(k) es negativo entonces kdu es zero
  u[2]=min_and(positivo(velocidadx),negativo(aceleracionx),1 0),   //DOF
  fi[2][0]=u[2];                                                    //u^d, corte de la funcion de salida
  //Regla 3 Si v(k) es negativo y dv(k) es cero  entonces kdu es negativo
  u[3]=min_and(negativo(velocidadx),zero(aceleracionx),1.0),        //DOF
  fi[3][1]=u[3],                                                    //u^d, corte de la funcion de salida
  //Regla 4 Si v(k) es cero y dv(k) es cero entonces kdu es cero
  u[4]=min_and(zero(velocidadx),zero(aceleracionx),1.0);            //DOF
  fi[4][1]=u[4];                                                    //u^d, corte de la funcion de salida
  //Regla 5 Si v(k) es positivo y dv(k) es cero entonces kdu es positivo
  u[5]=min_and(positivo(velocidadx),zero(aceleracionx),1 0);        //DOF
  fi[5][1]=u[5];                                                    //u^d, corte de la funcion de salida
  //Regla 6 Si v(k) es negativo y dv(k) es positivo entonces kdu es cero
  u[6]=min_and(negativo(velocidadx),positivo(aceleracionx),1 0);   //DOF
  fi[6][0]=u[6];                                                    //u^d, corte de la funcion de salida
  //Regla 7 Si v(k) es cero y dv(k) es positivo entonces kdu es positivo
  u[7]=min_and(zero(velocidadx),positivo(aceleracionx),1 0);        //DOF
```

```
 fi[7][0]=u[7],                                                    //u^d, corte de la funcion de salida
 //Regla 8 Si v(k) es positivo y dv(k) es positivo kdu es positivo
 u[8]=min_and(positivo(velocidadx),positivo(aceleracionx),1.0);    //DOF
 fi[8][2]=u[8];                                                    //u^d, corte de la funcion de salida
 //Calculo de la funcion de salida global, maximos
 for(i=0;i<9;i++) {
   if(fi[i][0]>=F[0])  F[0]=fi[i][0];
   if(fi[i][1]>=F[1])  F[1]=fi[i][1];
   if(fi[i][2]>=F[2])  F[2]=fi[i][2];
 }
 //defuzificacion y obtencion de la salida de control desnormalizada
 if((F[0]+F[1]+F[2])!=0)
   *gananciax=(int)(((-0.5*F[0]+0*F[1]+0.5*F[2])/(F[0]+F[1]+F[2]))*55.0+95 0),
 else                    // evita la division por cero
   *gananciax=145;       //ganancia por default

 //EVALUACION DE LAS REGLAS PARA Y
 F[0]=F[1]=F[2]=0 ;                                                //inicializacion de la funcion resultado a cero
 //Regla 0 Si u(k) es negativo y du(k) es negativo kdu es negativo
 u[0]=min_and(negativo(velocidady),negativo(aceleraciony),1.0);    //DOF
 fi[0][2]=u[0]; //u^d, corte de la funcion de salida
 //Regla 1 Si u(k) es cero y du(k) es negativo entonces kdu es negativo
 u[1]=min_and(zero(velocidady),negativo(aceleraciony),1 0),        //DOF
 fi[1][0]=u[1];                                                    //u^d, corte de la funcion de salida
 //Regla 2 Si u(k) es positivo y du(k) es negativo entonces kdu es zero
 u[2]=min_and(positivo(velocidady),negativo(aceleraciony),1.0);    //DOF
 fi[2][0]=u[2];                                                    //u^d, corte de la funcion de salida
 //Regla 3 Si u(k) es negativo y du(k) es cero kdu es negativo
 u[3]=min_and(negativo(velocidady),zero(aceleraciony),1.0);        //DOF
 fi[3][1]=u[3];                                                    //u^d, corte de la funcion de salida
 //Regla 4 Si u(k) es cero y du(k) es cero entonces kdu es cero
 u[4]=min_and(zero(velocidady),zero(aceleraciony),1.0);            //DOF
 fi[4][1]=u[4];                                                    //u^d, corte de la funcion de salida
 //Regla 5 Si u(k) es positivo y du(k) es cero kdu es positivo
 u[5]=min_and(positivo(velocidady),zero(aceleraciony),1.0);        //DOF
 fi[5][1]=u[5];                                                    //u^d, corte de la funcion de salida
 //Regla 6 Si u(k) es negativo y du(k) es positivo entonces kdu es cero
 u[6]=min_and(negativo(velocidady),positivo(aceleraciony),1.0);    //DOF
 fi[6][0]=u[6];                                                    //u^d, corte de la funcion de salida
 //Regla 7 Si u(k) es cero y du(k) es positivo entonces kdu es positivo
 u[7]=min_and(zero(velocidady),positivo(aceleraciony),1.0);                //DOF
 fi[7][0]=u[7];                                                            //u^d, corte de la funcion de salida
 //Regla 8 Si u(k) es positivo y du(k) es positivo kdu es positivo
 u[8]=min_and(positivo(velocidady),positivo(aceleraciony),1.0);            //DOF
 fi[8][2]=u[8];                                                            //u^d, corte de la funcion de salida
 //Calculo de la funcion de salida global
 for(i=0;i<9;i++) {
   if(fi[i][0]>=F[0])  F[0]=fi[i][0];
   if(fi[i][1]>=F[1])  F[1]=fi[i][1];
   if(fi[i][2]>=F[2])  F[2]=fi[i][2];
 }
 //defuzificacion y obtencion de la salida de control desnormalizada
 if((F[0]+F[1]+F[2])!=0)
   *gananciay=(int)(((-0.5*F[0]+0*F[1]+0.5*F[2])/(F[0]+F[1]+F[2]))*55.0+95.0);
 else
   *gananciay=145;                                                           //ganancia por default
}

/* Este metodo realiza el control difuso, recibiendo el valor del centroide actual
y devolviento los incrementos de control necesarios por coordenada, que incluyen
tanto el incremento en el control como la prediccion de velocidad. Actualiza
las variables de velocidad v1x y v1y para ser usadas por la funcion que calcula
la ganancia */

void Fuzzy_Logic_Controller::Accion_de_Control(int xo, int yo, int *dux, int *duy) {
  float wx=256.0,wy=240.0;                  //referencias para el control en x e y
  float ke=1.0/74;                          //constante de normalizacion para el error
```

```
float kd=1.0/135,                       //constante de normalizacion para la derivada del error
float kdu;                              //constante de normalizacion para la salida de control
float kv=1.0/64;                        //constante de normalizacion para la velocidad
float vel1x,vel1y;                      //velocidades en flotante normalizadas
float u[9];                             //DOF (degree of firing), son nueve reglas
float fi[9][3]={{0,0,0},
                {0,0,0},
                {0,0,0},
                {0,0,0},
                {0,0,0},
                {0,0,0},
                {0,0,0},
                {0,0,0},
                {0,0,0}};               //funciones de salida por regla,(singleton), son nueve reglas
float F[3]={0,0,0};                     //funcion de salida global
int i;                                  //contador
int ganx,gany;                          //registros de las ganancias a aplicar

//Centroides en coordenadas absolutas
pos_ant_x=pos_act_x; pos_ant_y=pos_act_y,       //actualiza centroide absoluto
pos_act_x=posx_logica_camara+xo;
pos_act_y=posy_logica_camara+yo,
//Velocidad del objeto
gotoxy(20,11); printf("                                 ");
gotoxy(20,11); printf("pos ant x %i  en y %i",pos_ant_x,pos_ant_y);
gotoxy(20,12); printf("                                 ");
gotoxy(20,12); printf("pos act x %i  en y %i",pos_act_x,pos_act_y);
gotoxy(20,13), printf("                                 ");
gotoxy(20,13); printf("pos cam x %i  en y %i",posx_logica_camara,posy_logica_camara),
v1x=pos_act_x-pos_ant_x;
v1y=pos_act_y-pos_ant_y;
gotoxy(20,14); printf("                                 ");
gotoxy(20,14), printf("v1 x %f  en y %f",v1x,v1y);
// obtiene las ganancias a aplicar para cada eje
ganancia_dinamica_FLC(&ganx,&gany);             //devuelve la ganancia a aplicar en x e y
gotoxy(20,15); printf("                                 ");
gotoxy(20,15); printf("ganancia x %i  en y %i",ganx,gany),
// parametro de simetria de las funciones de membresia
xs=0 5;
// parametro de variacion de positivo y negativo vel
//xv=-0.0,
// parametro de las exponenciales
alfa=1 0;
beta=8.0;
corrimiento=0.0,
beta_ganancia=0.0687;
//Actualizaciones
e1x=e2x; e1y=e2y;                               //actualiza error
e2x=wx-1.0*xo; e2y=wy-1.0*yo;                   //error actual realimentacion negativa
dex=e2x-(e1x/ke); dey=e2y-(e1y/ke),             //variacion del error desnormalizamos el error anterior
e2x=ke*e2x, e2y=ke*e2y;                         //normalizaciones
dex=kd*dex; dey=kd*dey;
vel1x=kv*v1x; vel1y=kv*v1y;

//EVALUACION DE LAS REGLAS PARA X
//Regla 0 Si e(k) es negativo y de(k) es negativo y v(k) is positivo entonces du es negativo
u[0]=min_and(negativo(e2x),negativo(dex),positivo_vel_e(vel1x));        //DOF
fi[0][0]=u[0],                                                          //u^d, corte de la funcion de salida
//Regla 1 Si e(k) es cero y de(k) es negativo entonces du es negativo
u[1]=min_and(zero(e2x),negativo(dex),1.0);                              //DOF
fi[1][1]=u[1];                                                          //u^d, corte de la funcion de salida
//Regla 2 Si e(k) es positivo y de(k) es negativo entonces du es zero
u[2]=min_and(positivo(e2x),negativo(dex),1.0);                          //DOF
fi[2][1]=u[2];                                                          //u^d, corte de la funcion de salida
//Regla 3 Si e(k) es negativo y de(k) es cero y v(k) es positivo entonces du es negativo
u[3]=min_and(negativo(e2x),zero(dex),positivo_vel_e(vel1x)),            //DOF
fi[3][0]=u[3];                                                          //u^d, corte de la funcion de salida
```

```
//Regla 4 Si e(k) es cero y de(k) es cero entonces du es cero
u[4]=min_and(zero(e2x),zero(dex),1.0);                          //DOF
fi[4][1]=u[4];                                                  //u^d, corte de la funcion de salida
 //Regla 5 Si e(k) es positivo y de(k) es cero y v(k) es negativo entonces du es positivo
u[5]=min_and(positivo(e2x),zero(dex),negativo_vel_e(vel1x));    //DOF
fi[5][2]=u[5];                                                  //u^d, corte de la funcion de salida
//Regla 6 Si e(k) es negativo y de(k) es positivo entonces du es cero
u[6]=min_and(negativo(e2x),positivo(dex),1.0);                  //DOF
fi[6][1]=u[6];                                                  //u^d, corte de la funcion de salida
//Regla 7 Si e(k) es cero y de(k) es positivo entonces du es positivo
u[7]=min_and(zero(e2x),positivo(dex),1.0);                      //DOF
fi[7][12]=u[7];                                                 //u^d, corte de la funcion de salida
//Regla 8 Si e(k) es positivo y de(k) es positivo y v(k) es negativo entonces du es positivo
u[8]=min_and(positivo(e2x),positivo(dex),negativo_vel_e(vel1x));//DOF
fi[8][2]=u[8];                                                  //u^d, corte de la funcion de salida
//Calculo de la funcion de salida global, maximos
for(i=0;i<9;i++) {
  if(fi[i][0]>=F[0])  F[0]=fi[i][0];
  if(fi[i][1]>=F[1])  F[1]=fi[i][1];
  if(fi[i][2]>=F[2])  F[2]=fi[i][2];
}
//defuzificacion y obtencion de la salida de control desnormalizada
kdu=(float)ganx;                                //ganancia calculada para X
if((F[0]+F[1]+F[2])!=0)
  *dux=(int)(((-0.5*F[0]+0*F[1]+0.5*F[2])/(F[0]+F[1]+F[2]))*kdu);
else                                            // evita la division por cero
  *dux=0;
*dux= -(*dux)+v1x;                              //control que incluye al FLC más la prediccion de velocidad
posx_logica_camara=posx_logica_camara-(*dux); //actualizacion de la posicion logica en x

//EVALUACION DE LAS REGLAS PARA Y
F[0]=F[1]=F[2]=0 ;                                              //inicializacion de la funcion resultado a cero
//Regla 0 Si e(k) es negativo y de(k) es negativo y v(k) es positivo entonces du es negativo
u[0]=min_and(negativo(e2y),negativo(dey),positivo_vel_e(vel1y));  //DOF
fi[0][0]=u[0];                                                  //u^d, corte de la funcion de salida
//Regla 1 Si e(k) es cero y de(k) es negativo entonces du es negativo
u[1]=min_and(zero(e2y),negativo(dey),1.0);                      //DOF
fi[1][10]=u[1];                                                 //u^d, corte de la funcion de salida
//Regla 2 Si e(k) es positivo y de(k) es negativo entonces du es zero
u[2]=min_and(positivo(e2y),negativo(dey),1.0);                  //DOF
fi[2][1]=u[2],                                                  //u^d, corte de la funcion de salida
//Regla 3 Si e(k) es negativo y de(k) es cero y v(k) es positivo entonces du es negativo
u[3]=min_and(negativo(e2y),zero(dey),positivo_vel_e(vel1y));    //DOF
fi[3][0]=u[3];                                                  //u^d, corte de la funcion de salida
//Regla 4 Si e(k) es cero y de(k) es cero entonces du es cero
u[4]=min_and(zero(e2y),zero(dey),1.0);                          //DOF
fi[4][1]=u[4];                                                  //u^d, corte de la funcion de salida
//Regla 5 Si e(k) es positivo y de(k) es cero y v(k) es negativo entonces du es positivo
u[5]=min_and(positivo(e2y),zero(dey),negativo_vel_e(vel1y));    //DOF
fi[5][2]=u[5];                                                  //u^d, corte de la funcion de salida
//Regla 6 Si e(k) es negativo y de(k) es positivo entonces du es cero
u[6]=min_and(negativo(e2y),positivo(dey),1.0);                  //DOF
fi[6][1]=u[6];                                                  //u^d, corte de la funcion de salida
//Regla 7 Si e(k) es cero y de(k) es positivo entonces du es positivo
u[7]=min_and(zero(e2y),positivo(dey),1.0);                      //DOF
fi[7][12]=u[7];                                                 //u^d, corte de la funcion de salida
//Regla 8 Si e(k) es positivo y de(k) es positivo y v(k) es negativo entonces du es positivo
u[8]=min_and(positivo(e2y),positivo(dey),negativo_vel_e(vel1y));//DOF
fi[8][2]=u[8];                                                  //u^d, corte de la funcion de salida
//Calculo de la funcion de salida global
for(i=0;i<9;i++) {
  if(fi[i][0]>=F[0])  F[0]=fi[i][0];
  if(fi[i][1]>=F[1])  F[1]=fi[i][1];
  if(fi[i][2]>=F[2])  F[2]=fi[i][2];
}
//defuzificacion y obtencion de la salida de control desnormalizada
kdu=(float)gany;                                                //ganancia aplicada en Y
```

```
  if((F[0]+F[1]+F[2])!=0)
    *duy=(int)(((-0.5*F[0]+0*F[1]+0.5*F[2])/(F[0]+F[1]+F[2]))*kdu),
  else
    *duy=0;                                          //evita la division por cero
  *duy= -(*duy)+v1y,                                 //control que incluye al FLC más la prediccion de velocidad
   posy_logica_camara=posy_logica_camara-(*duy);  //actualizacion de la posicion logica en y
}


/*-----------------------------Programa principal-----------------------------------*/
void main(void) {
   long cx, cy,            // centroide de la imagen
   int dux,duy,            // acciones de control
   float pasosx,pasosy;      // pasos de control
   clrscr();
   puerto_4f0h.calibra();
   printf("\npase  calibra pulsa tecla para continuar");
   while(!kbhit()),
   getc(stdin);
   do {
     imagen.obten_centroide(30,&cx,&cy);
     gotoxy(20,10), printf("                                          ");
     gotoxy(20,10); printf("centroide x %li  en y %li",cx,cy);
     controlador.Accion_de_Control((int) cx, (int) cy, &dux, &duy),
     gotoxy(20,16); printf("                                          ");
     gotoxy(20,16); printf("control x %d  en y %d ",dux,duy);
     puerto_4f0h.convierte_control_de_pixeles_a_pasos(dux,duy,&pasosx, &pasosy);
     gotoxy(20,17); printf("                                          ");
     gotoxy(20,17); printf("control pasos x %f en y %f ",pasosx,pasosy),
     gotoxy(20,18); printf("                                          "),
     gotoxy(20,18), printf("mueve en x %f grados en y %f grados", (pasosx/5 0)*1.8,(pasosy/5 0)*1.8);
     puerto_4f0h posiciona_motores(pasosx,pasosy),
     while ( puerto_4f0h.mueve_motores(2,0,0,0)==0);
     puerto_4f0h.retardo(4000L); //////quitarlo es sólo para depuracion
   } while(!kbhit());
}
```