ANEXO A:
MODELO DE DESCRIPCIÓN DE LA SOLUCIÓN PARA EL PROTOTIPO DEL SISTEMA DE MEDICIÓN DE DESPLAZAMIENTO EN DOS EJES BASADO EN TÉCNICAS DE VISIÓN ARTIFICIAL

ZULMA BERNARDA PABÓN PIPICANO
ANCIZAR SANTACRUZ AHUMADA

Director
I. E. Juan Fernando Flórez Marulanda

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRÓNICA, INSTRUMENTACIÓN Y CONTROL
POPAYÁN
2007

ANEXO A: MODELO DE DESCRIPCIÓN DE LA SOLUCIÓN M. D. S.

A.1. ANÁLISIS DE REQUERIMIENTOS

El campo de la visión artificial ofrece muchas opciones a la hora de desarrollar aplicaciones de inspección, monitoreo y medida. No obstante, los sistemas de visión artificial son, en general, ubicados en espacios cerrados, con ambientes controlados de iluminación y presencia de objetos en la escena. En el caso presentado en este modelo de solución el sistema va a estar ubicado en un espacio externo con condiciones ambientales que no se pueden controlar ni predecir.

El primer parámetro a tener en cuenta es la definición de las características del entorno en cuanto a:

- Las características de interés del objeto medido.
- Las condiciones de la escena o plano donde se ubica el objeto.
- Las condiciones de los objetos de entorno vistos por detrás o junto al objeto (fondo).
- El tipo de ambiente del sistema (externo o interno) y las características asociadas.
- El uso de un sistema de iluminación adecuado a las características del objeto a inspeccionar, la escena o plano visto por el sensor y las características del sensor.
- Las características de desempeño del sensor y las lentes asociadas.
- La razón de cambio de los objetos entre imágenes sucesivas.

El segundo parámetro corresponde a las condiciones de desempeño del sistema de medición en conjunto, especialmente las referidas al motor de procesamiento de imágenes. Entre estas características se encuentran:

- La resolución y profundidad de color del sensor.
- El tamaño de la memoria de alojamiento de las imágenes de entrada.
- El tamaño de la memoria para el procesamiento de datos.
- La velocidad de lectura/ escritura de los datos.
- La velocidad de procesamiento de cada imagen.

Por último, se deben tener en cuenta las condiciones referidas al desempeño esperado por el sistema en cuanto a:

- La precisión de la medida entregada.
- La velocidad de respuesta.
- El ancho de banda de la señal de salida (correspondencia con la razón de cambio de la escena).

A.1.1. Condiciones del entorno

El entorno de trabajo del sistema propuesto es un ambiente externo donde se encuentre una estructura a la cual se le ha de medir la deflexión en algún punto específico, por ejemplo, debajo de un puente metálico. En todo ambiente externo las condiciones son altamente cambiantes, más aún en las regiones de la geografía del País. El sistema va a estar expuesto a lluvia, viento y cambios de temperatura y para protegerlo de estas condiciones se requiere ubicar un punto de instalación del sistema tal que la estructura pueda resguardar a los dispositivos y módulos electrónicos.

El sistema de visión va a estar expuesto a fuertes cambios de iluminación que suceden por el cambio en la posición del sol durante el día y el brillo de este debido a la nubosidad presente. También afectan los relámpagos que se pueden producir en la zona, pues estos agregan una fuerte cantidad de luz en instantes cortos y esporádicos hasta tal punto de superponerse totalmente a la iluminación artificial definida para la observación por el sistema de visión. Por este motivo la fuente de iluminación debe ser intensa, constante y debe ofrecer el más alto contraste al objeto a medir en condiciones adversas.

El campo de visión de la cámara probablemente estará compuesto de una variedad de elementos de diversos colores, unos constantes como la vegetación y los elementos de la estructura y otros esporádicos como insectos, partículas de polvo, gotas de lluvia, aves o vehículos. En la mayoría de los casos los objetos presentes son opacos y difusos, en el caso de la vegetación predomina el color verde y en el caso de los elementos de la estructura predominan los colores grises y el amarillo. No obstante, los vehículos pueden llevar luces encendidas y estas pueden ser blancas, azulosas, amarillas o rojas. Las luces

de los vehículos se mezclarán con las demás fuentes que aporten iluminación a la escena y cambiarán los colores de los elementos presentes.

El sistema requiere ser instalado en lugares de difícil acceso donde generalmente se encuentran tanto el punto de referencia para la ubicación de la cámara como el punto de deflexión a medir. Esta dificultad a la hora de la instalación es una condición de protección contra intrusos y vandalismos pero limita el acceso posterior para hacer mantenimiento o calibración.

Si la estructura es un puente carretero metálico, experimentará vibraciones mecánicas debidas a diversas fuentes como el viento, los sismos de la zona o el paso de vehículos, personas o animales con amplitudes que están entre menos de un milímetro y hasta los 10 centímetro. Las vibraciones pueden ser de más de la decena de hertz y se propagarán por todos los elementos hasta llegar a las bases y los pilotes o de baja frecuencia sobre todo cuando pasa un vehículo de gran peso por la superestructura. El sistema debe estar en capacidad de rechazar las vibraciones rápidas a la vez que observa el desplazamiento de los elementos por las vibraciones de baja frecuencia, este desplazamiento es una ondulación con frecuencias por debajo de los 2 Hz.

A.1.2. Características del objeto a medir

Al momento de procesar imágenes en un sistema de visión artificial se debe definir previamente las características que diferencian e identifican el objeto a medir, ya sea que este se presente en forma espontánea en la escena o que aparezca en una secuencia de imágenes con cambios en algún parámetro. Para medir un desplazamiento o deformación en una estructura, se puede tomar como objeto a medir algún componente de la misma, los cuales varían en forma y dimensiones de una estructura a otra y en cuyo caso el sistema de procesamiento debe ser capaz de adaptarse cada vez que se instale el sistema de medición o cada vez que el componente observado cambie alguno de sus parámetros que lo identifican. Otra forma es adherir a la estructura un elemento previamente conocido por el sistema de visión, cuyos parámetros identificadores permanezcan constantes en el tiempo, se pueda diferenciar del fondo de la imagen y que

transfiera las variables y comportamiento de la estructura a la imagen; dicho elemento tiene que ser solidario a la estructura en el punto de medida.

Cuando se escoge un objeto externo para medir un punto en la estructura, se requiere que dicho objeto permita ser identificado con claridad, entregue datos del comportamiento de la estructura de manera precisa y confiable y que no cambie con el tiempo. De la misma manera se requiere que sea acorde con el sistema de iluminación bajo cualquier condición ambiental y que pueda ser diferenciado y extraído del fondo de la imagen.

A.1.3. Características del sistema de procesamiento

Para construir el sistema se requiere una cámara de video a color de alta resolución y buena velocidad de captura. Se espera que a mayor resolución haya mayor precisión porque hay más datos para procesar la imagen y porque se pueden captar detalles más finos. A mayor velocidad de captura se puede medir vibraciones de mayor frecuencia porque se aumenta la frecuencia de muestreo para cada uno de los ejes, siguiendo el teorema de Nyquist.

Entre más resolución tenga la imagen se debe tener procesadores más potentes porque tendrán que analizar una cantidad mayor de datos por imagen y de la misma manera ocurre al incrementar la velocidad de captura porque el tiempo de procesamiento entre un cuadro y otro se reduce. Se puede partir de una resolución superior a VGA porque se podría captar desplazamientos de alrededor de una milésima del rango y velocidades de captura de más de 10 cuadros por segundo para garantizar que se capte las ondulaciones principales de la deformación elástica de la estructura sin problemas de alias.

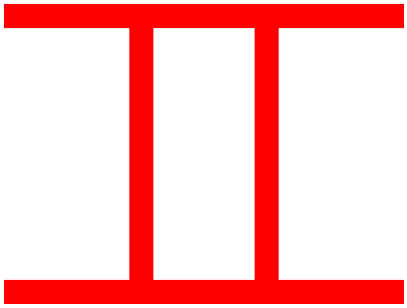
Un procesador común puede llegar a realizar más de 1000 millones de cálculos por segundo y un DSP puede alcanzar los 8000 millones. Como se requiere procesar una enorme cantidad de datos a una alta velocidad es mejor generar un algoritmo para operar sobre un DSP. No obstante, el algoritmo que se diseñe debe ser lo más liviano computacionalmente a la vez que confiable y efectivo

A.2. DISEÑO DEL SISTEMA DE MEDICIÓN DE DESPLAZAMIENTO

A.2.1. Objeto de medida

Para el diseño del algoritmo se ha escogido adherir un elemento externo plenamente conocido por el sistema. El objeto tiene una figura geométrica simple y bien definida, colores e iluminación controlada suministrada por un juego de luces rojas y dimensiones claramente establecidas adecuadas al rango y resolución de la medida. La figura se muestra en la ilustración 1.

Ilustración 1. Forma patrón del objeto a medir por el sistema



La forma surge de la necesidad de tener una grilla de calibración tan simplificada como sea posible, que a la vez sea confiable y ofrezca información certera al sistema de visión. Esta forma ofrece al sistema de visión la suficiente información para determinar su posición dentro de la imagen y su conversión a unidades físicas. A partir de las intersecciones, esquinas, posición del recuadro que contiene la imagen y el centro de masa de la figura se puede determinar la posición de esta en la imagen. El centro de masa de la figura, por su simetría, se encuentra en el centro del recuadro que la contiene y se puede hallar en la intersección de las diagonales del rectángulo interno o con el promedio de las posiciones de las intersecciones.

A partir de las dimensiones de la figura dadas en píxeles en cuanto a distancias, longitudes y tamaños de la imagen previamente interpretados como dimensiones físicas se puede calibrar el sistema para traducir la posición del objeto en píxeles a unidades físicas de ingeniería.

Al observar la pendiente de las líneas horizontales y/o verticales se puede determinar el giro de la figura en la imagen en grados o radianes en un rango (-90º, 90º).

Debido al paralelismo y la figura basada en rectas se puede determinar los fenómenos ópticos de aberración de cojín y aberración de barril debido a las lentes. Una vez determinado esto se puede iniciar un proceso de reconstrucción de la imagen.

El color rojo ha sido escogido por ser claramente diferenciable del entorno al contener elementos poco comunes en el fondo y porque se puede registrar en un solo plano de la imagen usando una cámara de video a color, lo que simplifica la separación del objeto del fondo al aplicar un filtro de color sobre la imagen que deje sólo los elementos totalmente rojos y elimine aquellos de otros colores.

A.2.2. Diseño del algoritmo de procesamiento

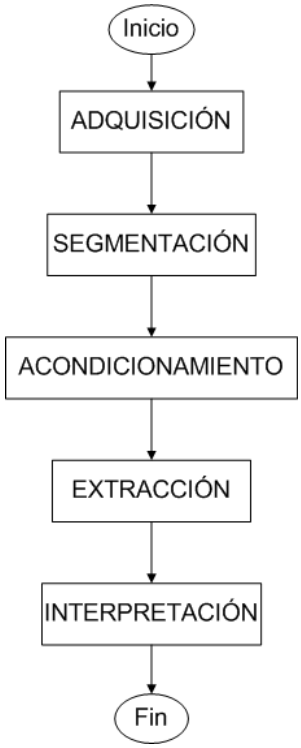
Para el desarrollo del algoritmo se utilizó una metodología de explosión en burbuja. Primero se probaron diferentes métodos de obtención de coordenadas como la medida de centroide, el código de cadena sobre el borde para detectar esquinas, la búsqueda de patrones o la transformada de hough para líneas. Al final, una mezcla de estas técnicas resultó en el modelo que se describe en este documento.

El algoritmo para procesamiento de imágenes se compone de diferentes fases que inician con la adquisición hasta llegar a la interpretación de los datos obtenidos de la imagen. Cada fase se realiza una vez por cada imagen y se repite indefinidamente mientras el sistema esté en funcionamiento y haya nuevas imágenes por procesar. Las fases necesarias para procesar la imagen se muestran en la ilustración 2. A su vez cada fase está compuesta por varios métodos y funciones que se irán explotando en las explosiones siguientes, partiendo del programa principal hasta llegar a las funciones de procesamiento.

La fase de adquisición tiene que ver con el almacenamiento de la imagen proveniente de la cámara en una memoria externa; esa imagen inicial contiene tres bytes por píxel (uno por cada color) que luego se condensa en una imagen en escala de grises. En la fase de

segmentación se procura extraer el objeto del fondo luego de analizar el histograma y obtener un umbral. Luego de esta fase se obtiene una imagen binaria de tal manera que se puede almacenar un píxel en cada bit para posteriormente procesar morfológicamente al aplicar operadores AND y OR entre vecinos de cada píxel.

Ilustración 2. Fases del algoritmo de procesamiento de imágenes

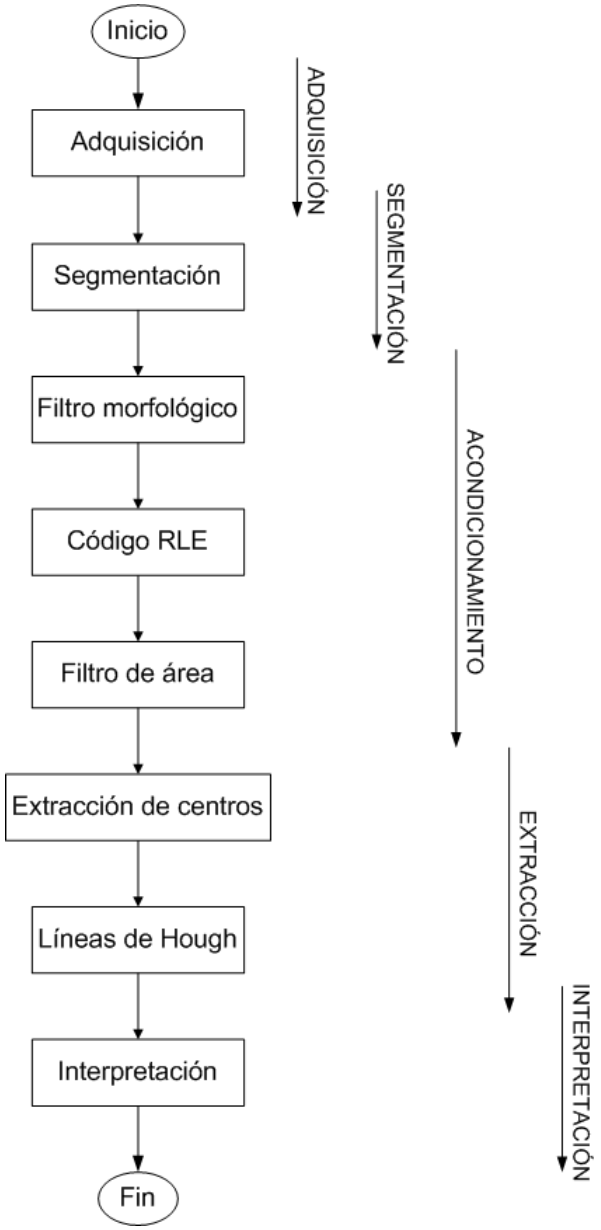


Una vez que la imagen se encuentra acondicionada pasa a una fase de extracción de características donde se etiquetan y miden los objetos para determinar su pertenencia al objeto a medir y su posición dentro de la imagen al combinar diversas técnicas. Por último se debe interpretar las medidas para verificar que correspondan verdaderamente a la traslación provocada por la estructura y se entregue el resultado en unidades físicas.

El programa principal se encuentra descrito por el diagrama de flujo de la ilustración 3. Aquí se observa que el procesamiento es secuencial y que las fases se combinan para permitir una mejor definición de los procesos planteados. Esto significa que cada acción

realizada complementa la anterior y prepara la imagen para la siguiente, de esta manera se agiliza el proceso y se simplifican los pasos.

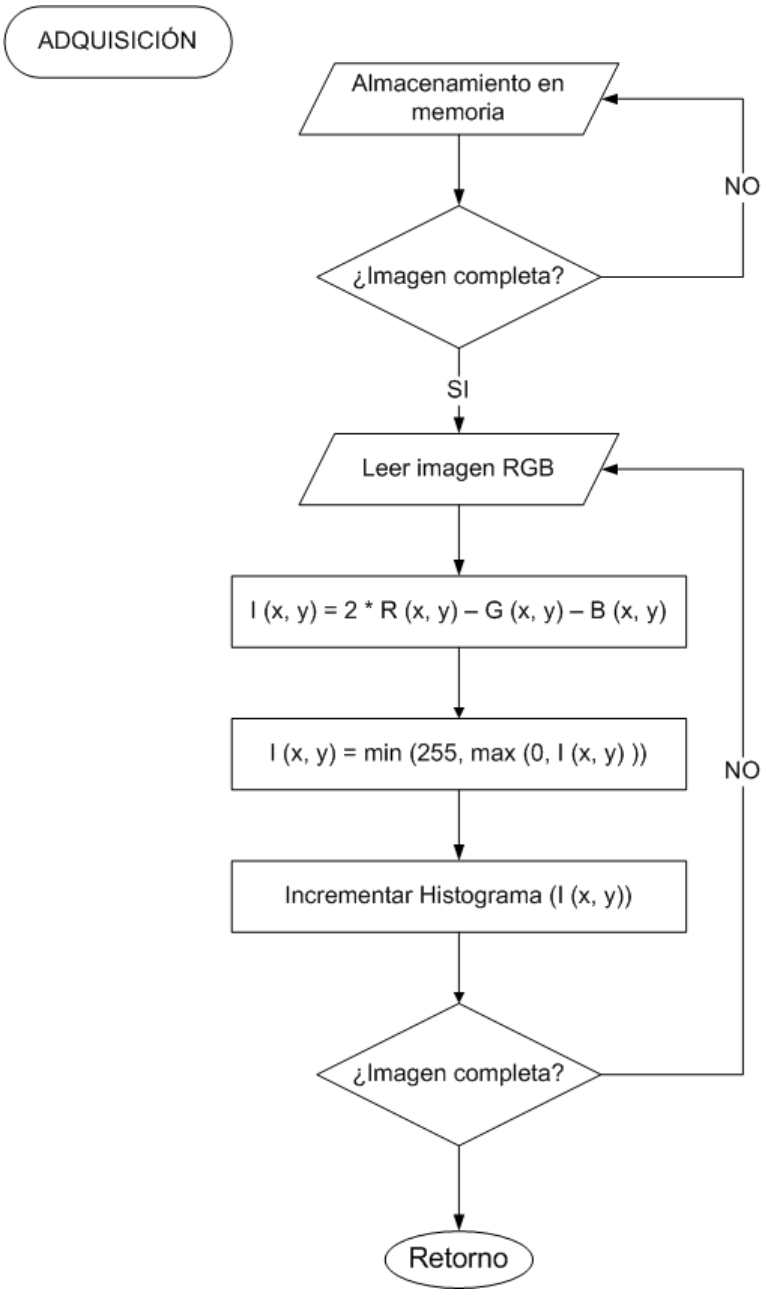
Ilustración 3. Diagrama de flujo del programa principal



El programa está compuesto del llamado a ocho funciones principales que a su vez pueden comprender otras funciones. Estas funciones se describen con mayor detalle en los diagramas siguientes.

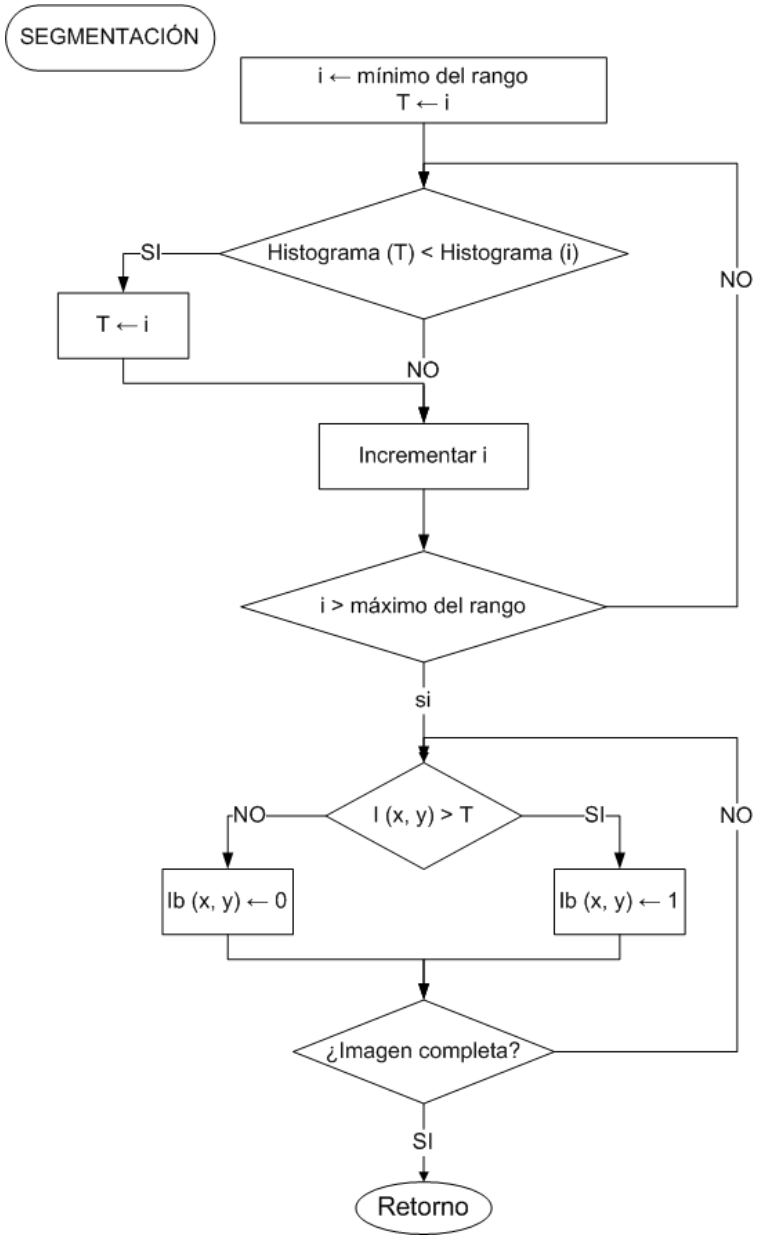
En la función de *Adquisición* se abre la comunicación con la pasarela y se leen los datos que llegan hasta completar la imagen en los tres planos de color. Luego, se aplica un filtro de color que realza los elementos que son enteramente rojos en la imagen y atenúa los elementos que contienen otros colores. Esto hace que se reduzca a un solo plano la imagen y de ella se puede obtener el histograma, como se muestra en la ilustración 4.

Ilustración 4. Función de adquisición de una imagen filtrada por color



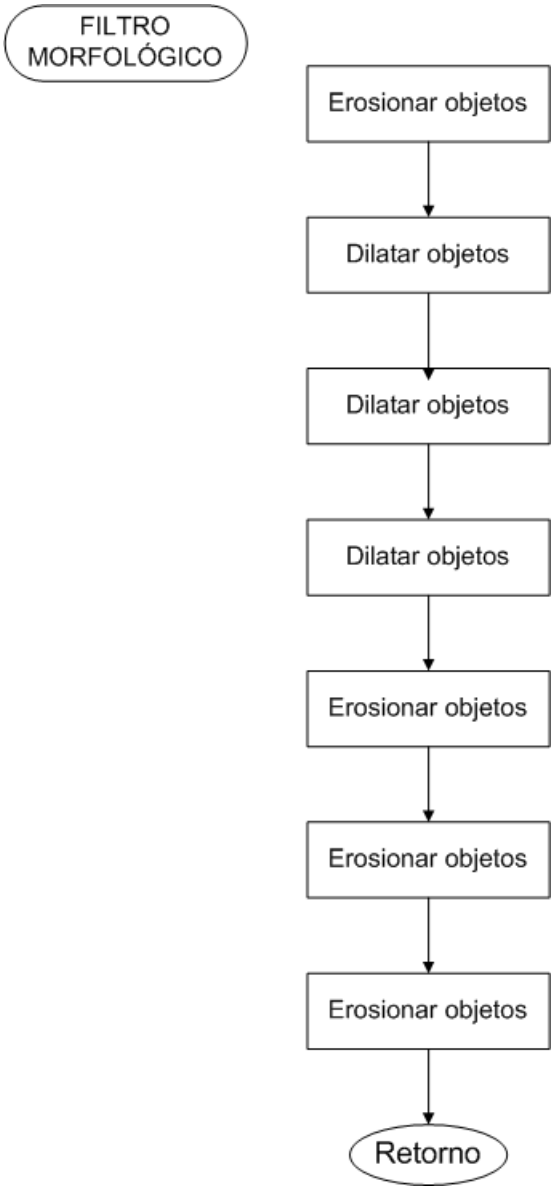
En la función de Segmentación se calcula el umbral al recorrer el histograma en un rango predefinido en compilación mediante el índice i. Cada vez que se halle un mínimo, se almacenará la posición en T. Luego, se compara cada elemento de la imagen con T y se asigna 0 a los elementos que no lo superan y 1 a los que si lo hacen, hasta completar toda la imagen. De esta manera quedan separados los elementos del fondo (ceros) y los elementos de los objetos (unos). Este método se aprecia en la ilustración 5.

Ilustración 5. Función de segmentación de una imagen binaria



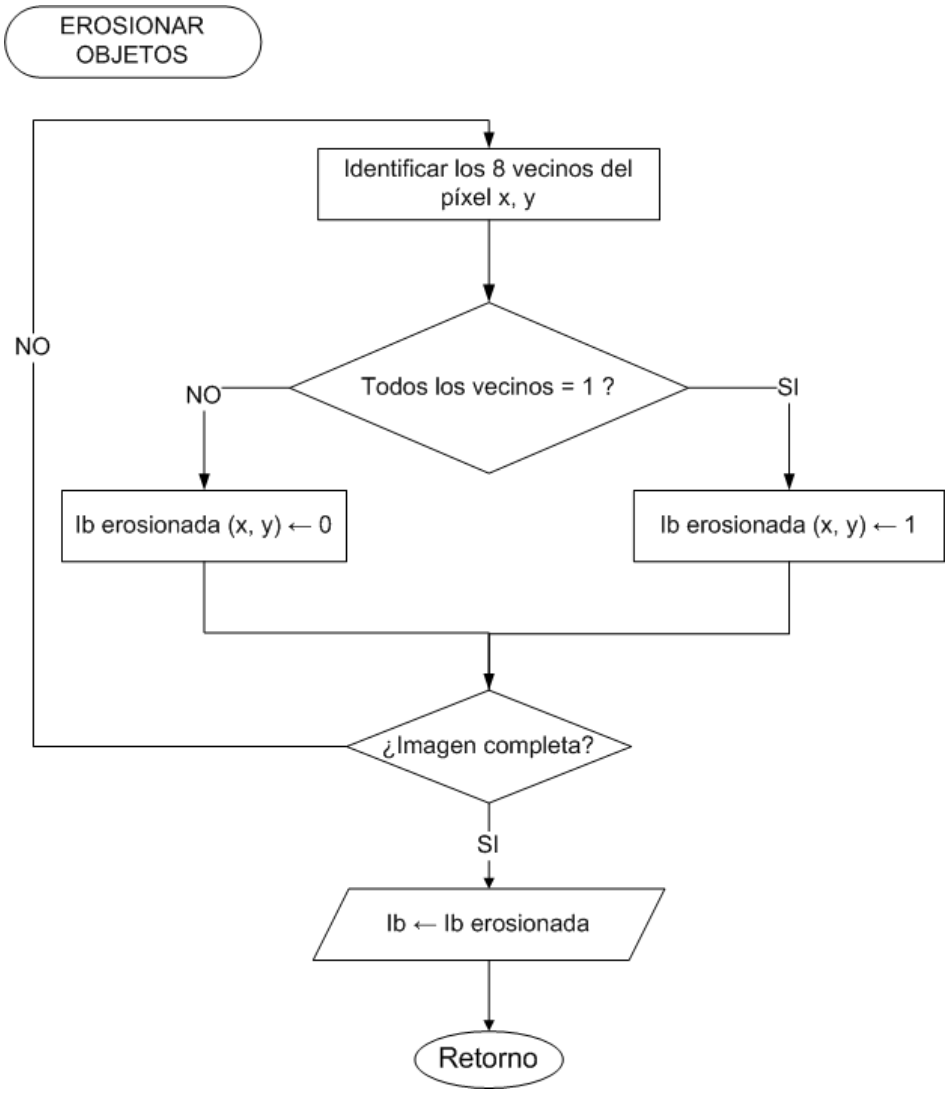
En la función de *Filtro Morfológico* se aplican varias etapas de erosión al realizar operaciones AND sobre los vecinos de cada píxel y luego, dilataciones al realizar operaciones OR sobre esos vecinos, complementado por nuevas erosiones, como se muestra en la ilustración 6. Para mayor detalle puede apreciarse una vista con la explosión del siguiente nivel de las funciones de erosión y dilatación, en las ilustraciones 7 y 8.

Ilustración 6. Función de filtro morfológico por combinación de erosiones y dilataciones



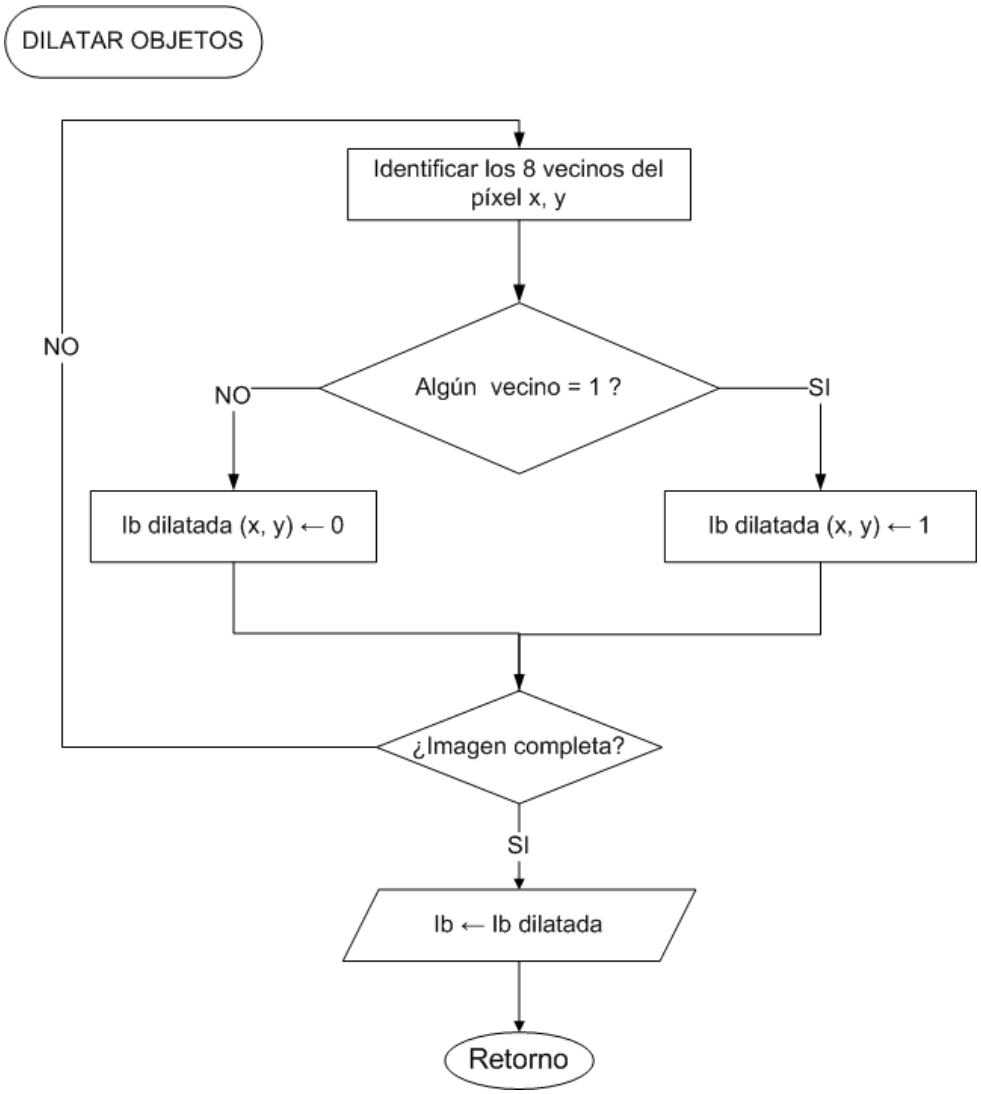
En la función *Erosionar Objetos* se hace una operación lógica AND entre todos los vecinos de cada píxel hasta completar la imagen. Si todos los vecinos son '1' entonces, el píxel de la imagen erosionada también lo será. En caso contrario será '0'. Este proceso se realiza recibiendo una imagen de entrada Ib y definiendo una imagen de salida "Ib erosionada" que al final actualiza la de entrada. La función se describe en la ilustración 7.

Ilustración 7. Función de erosión de una imagen binaria



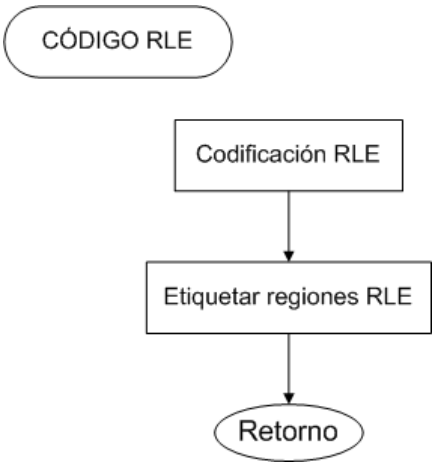
De manera análoga a la función de erosión, en la función *Dilatar Objetos* se realiza una operación lógica OR entre los vecinos de cada píxel de la imagen hasta recorrerla enteramente. Esto se puede apreciar en la ilustración 8.

Ilustración 8. Función de dilatación de una imagen binaria



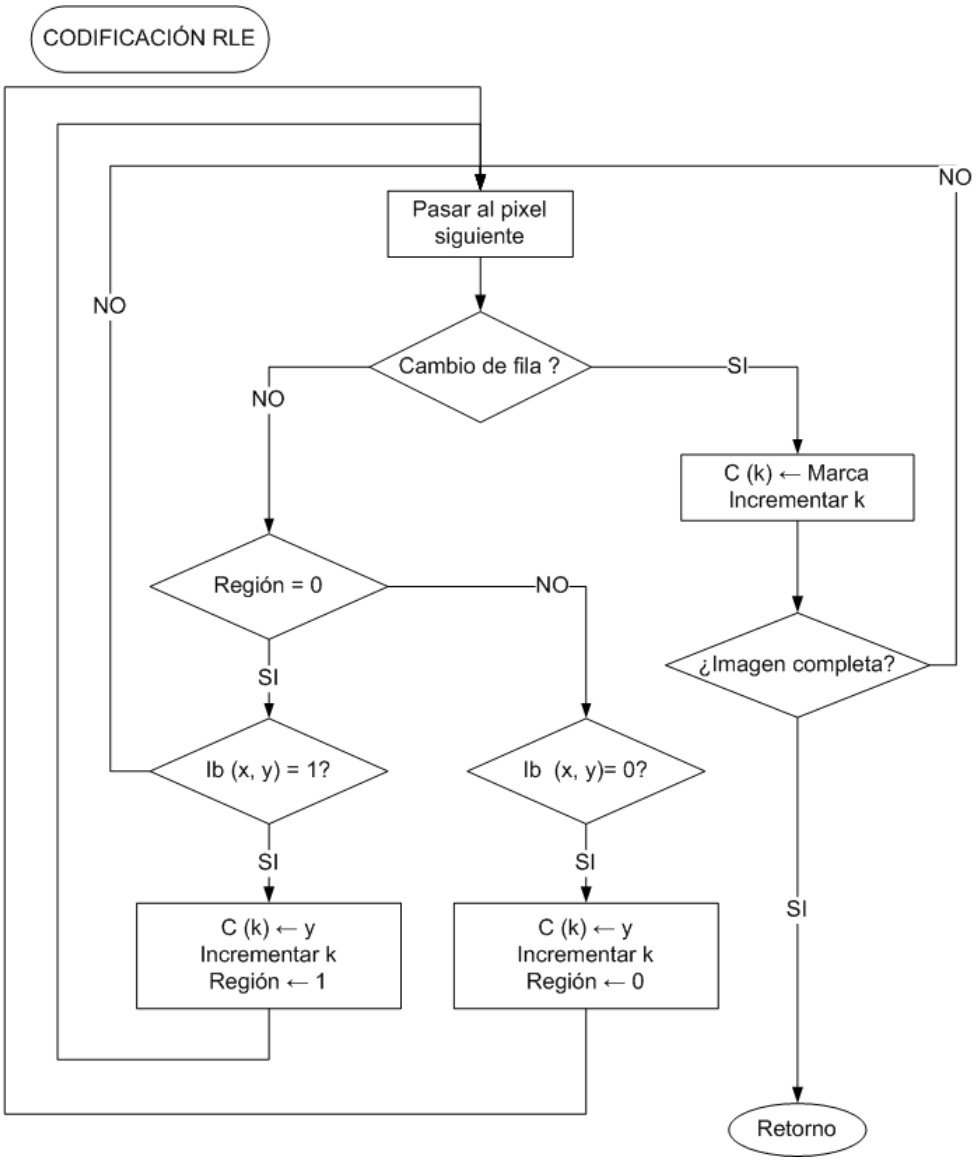
Después de filtrado el objeto se obtendrá una imagen con remoción de los objetos más pequeños y el resto con los bordes suavizados y sin huecos. Entonces, la imagen binaria pasa a una función de obtención del *código RLE* donde se realiza una compresión sin pérdidas. Una vez comprimida la imagen se puede etiquetar las regiones que describe el código para especificar su conexión entre ellas y de esta manera agruparlas todas hasta establecer e identificar todos lo objetos presentes. En las ilustración 9 se muestra el diagrama de flujo que luego se explosiona en las ilustraciones 10 y11 para mostrar mayores detalles

Ilustración 9. Función de generación del código RLE y su etiquetamiento



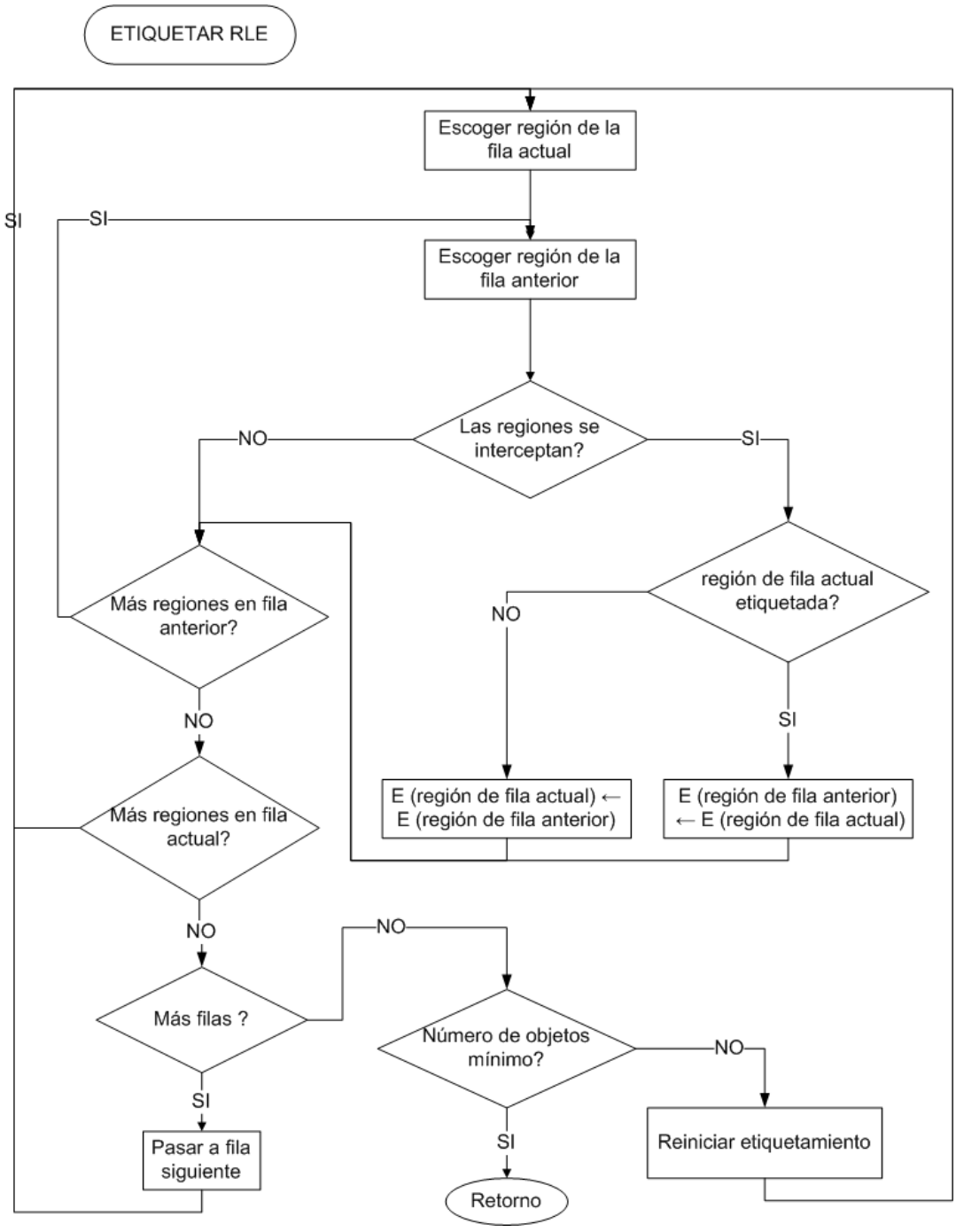
En la función *Codificación RLE* se hace una compresión de la imagen binaria y se almacena en un vector denominado C como se muestra en la ilustración 10. En esta función cada vez que se encuentra un cambio de 0 a 1 entre píxel y píxel, se registra la columna y se cambia el estado de la región presente; de la misma manera ocurre cuando encuentra un cambio de 1 a 0, solamente que se registra la última columna donde encontró un 1. Para tener un control sobre las filas, se agrega una marca de cambio de fila al pasar a la siguiente; esta marca puede ser un número que no corresponda a ninguna columna posible.

Ilustración 10. Función de compresión por codificación RLE



En la función *Etiquetar regiones RLE* se recibe el código RLE y se le asigna una etiqueta a cada región que este contenga considerando la conectividad con las regiones de la fila anterior. Las etiquetas se almacenan en un vector llamado E. Una aproximación del algoritmo de etiquetamiento se observa en la ilustración 11.

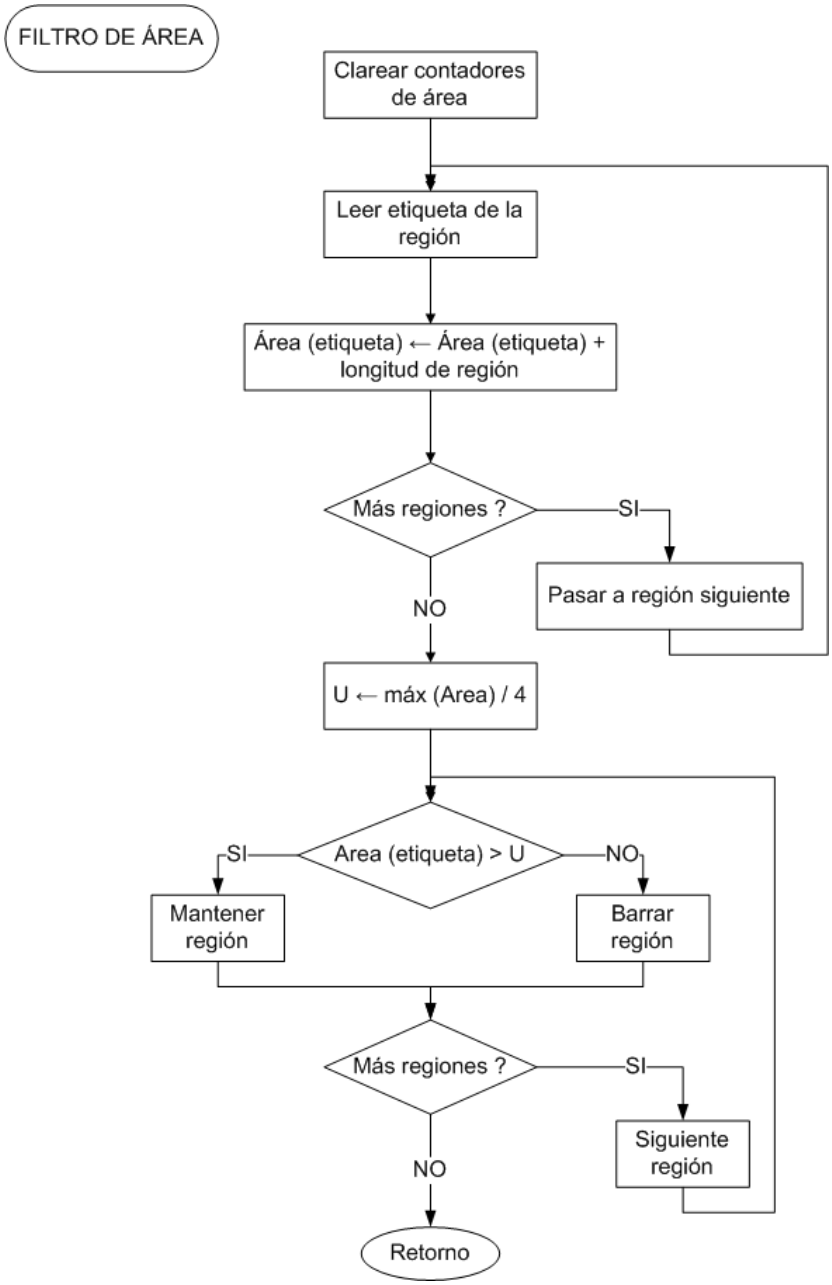
Ilustración 11. Algoritmo de etiquetamiento basado en codificación RLE



En la función *filtro de Área* primero se determina el área de cada objeto, al sumar las longitudes de cada una de las regiones del código RLE que conforman un objeto, según el vector de etiquetas asociado. Una vez hecho esto, se escoge el valor máximo de área y

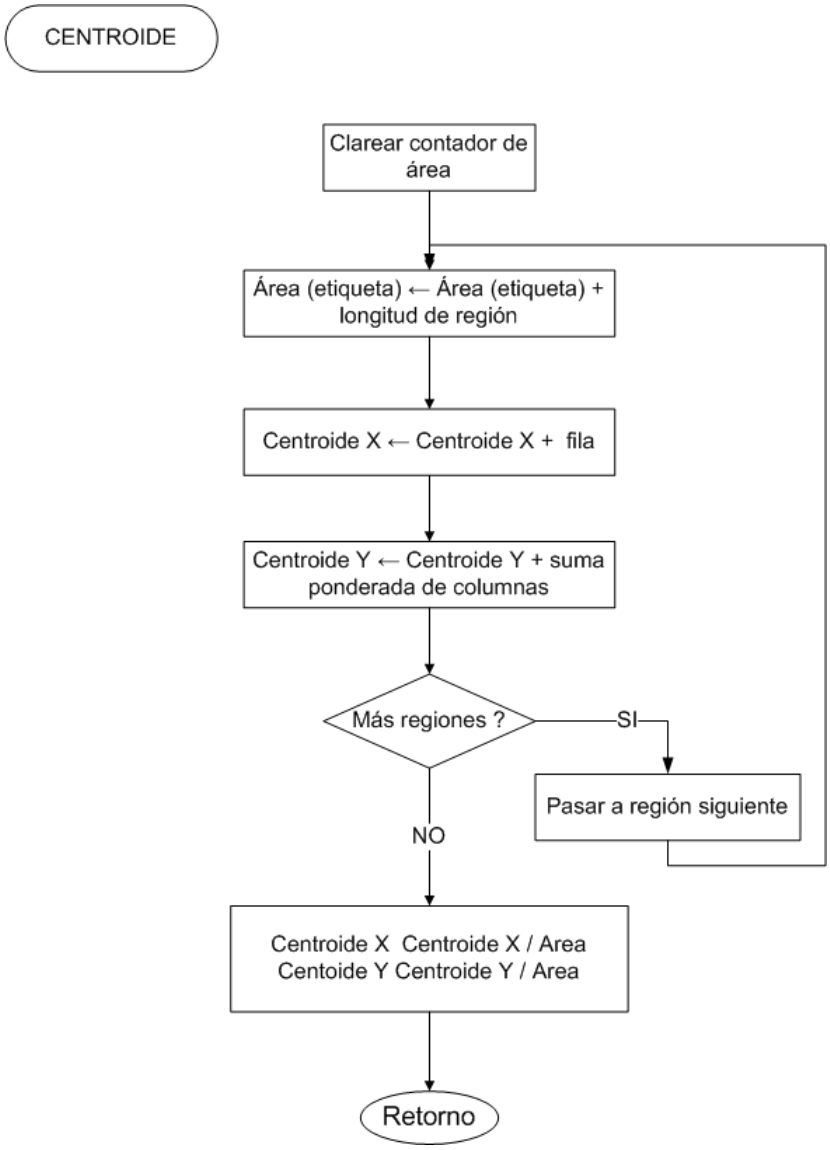
se fracciona para determinar un umbral. Luego se escoge aquellas regiones del código que pertenecen a objetos cuya área supera el umbral, el resto se borra del código. Una breve descripción de este método se muestra en la ilustración 12.

Ilustración 12. Función de remoción de objetos pequeños por código RLE



La función *extracción de centros* comprende las funciones de medida de centroide y obtención de una ROI. La función *centroide* calcula el centroide de la imagen asumiendo que todos los objetos componen uno solo. Para medirlo, recorre todas las regiones del código y calcula el área al acumular longitudes a la vez que acumula las filas y las columnas de las regiones. Al final, divide los acumuladores de centroide entre el área, como se muestra en la ilustración 13.

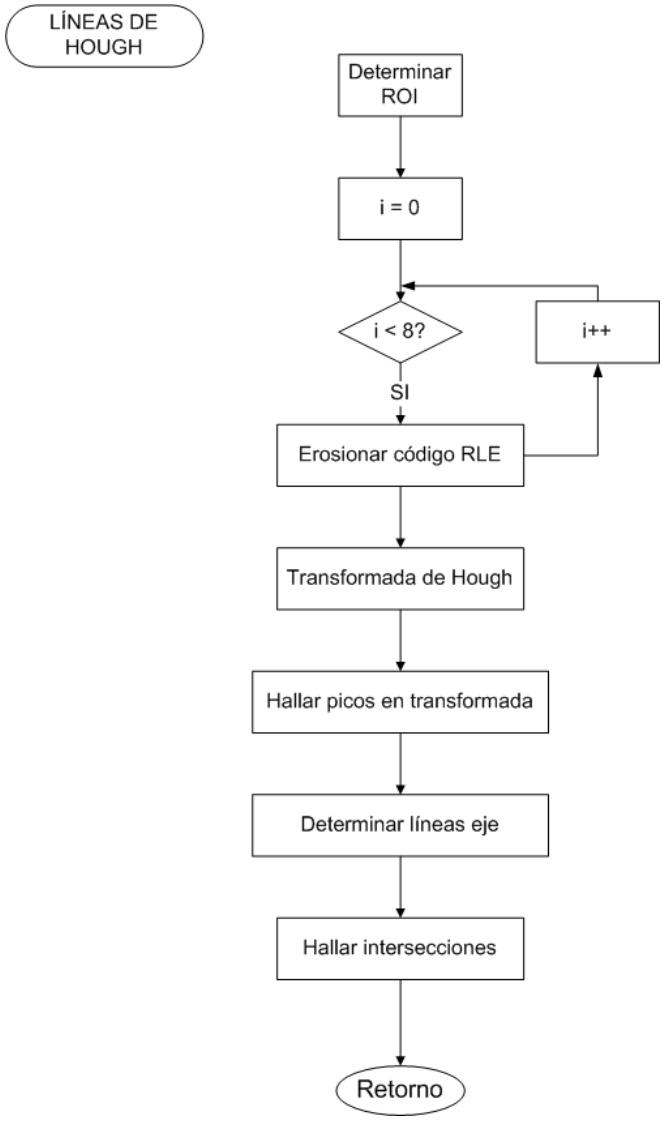
Ilustración 13. Función para medida del centroide sobre RLE



Por su parte la función *obtención de ROI* recorre todo el código y encuentra el menor de todos los orígenes de región, el mayor de todos los finales y la primera y última filas con una región en el código. El cruce de estas columnas dan origen a cuatro puntos y el promedio de esos puntos es el centro de ROI.

Luego de eso se opera sobre la transformada de Hough de la imagen, como puede apreciarse en la ilustración 14.

Ilustración 14. Función de obtención de ejes por transformada de Hough



Primero se erosiona el objeto varias veces para acentuar la forma de los ejes. Luego se halla las líneas representativas mediante la transformada de líneas, la cual entrega una matriz de acumulación con el voto de todos los píxeles que pueden pertenecer a una línea. Se escoge los cuatro máximos de esta matriz y sus coordenadas representan los parámetros de recta de los cuatro ejes que al intersectarlos originan cuatro puntos. Las distancias entre los puntos permiten la calibración y el promedio de ellos es un centro similar al centro de ROI o al centroide.

Por último se establece cual de las medidas tomadas representa realmente el centro del objeto y para ello se tiene en cuenta el número de objetos después del filtro de área y el número de ejes obtenidos.

A.3. IMPLEMENTACIÓN DEL SISTEMA PROTOTIPO

A.3.1. Construcción de un módulo de luces de referencia

El módulo de luces de referencia es un objeto emisivo que internamente tiene incorporado el sistema de iluminación. Para construirlo, primero se diseño un circuito electrónico donde se alinee un conjunto de LEDs de chorro rojo alimentados por una fuente DC. Para construir la figura deseada se requirieron cuatro tarjetas, una por cada segmento de la figura a crear. El circuito consta de un conjunto de resistencias limitadoras y varios diodos que permiten la conexión de una fuente múltiple para que haya redundancia. La fuente DC está basada en reguladores integrados que junto a un transformador, un puente rectificador y algunos condensadores convierte el voltaje de red (110 VAC) en un voltaje 5VDC.

En la ilustración 15 puede apreciarse el circuito de la tarjeta de 5 centímetros formada por 10 LEDs que se utiliza para formar los segmentos verticales; en la ilustración 16 se puede apreciar el circuito de la tarjeta de 8 centímetros formada por 16 LEDs que se utiliza para formar los segmentos horizontales y en la ilustración 17 se muestra la fuente de alimentación con redundancia.

Ilustración 15. Esquema electrónico de las tarjetas de 5 cm

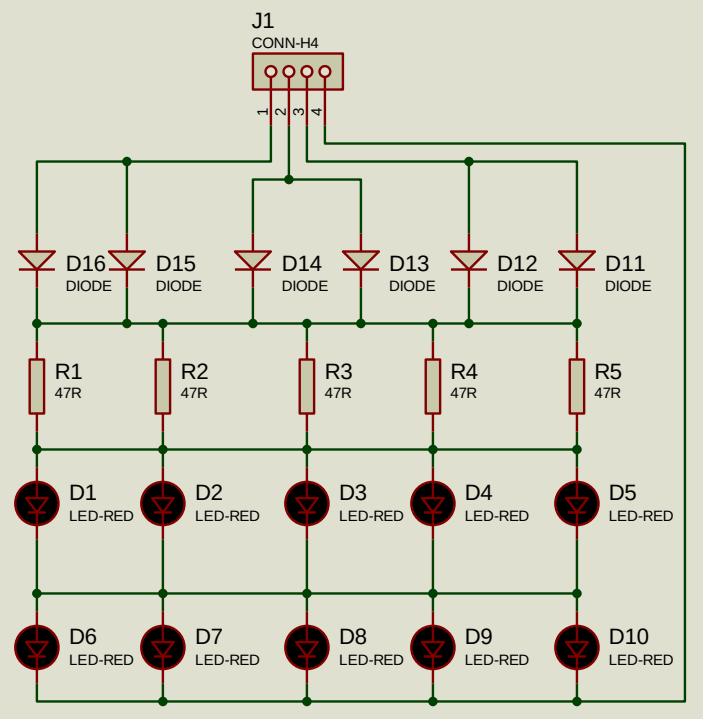


Ilustración 16. Esquema electrónico de las tarjetas de 8 cm.

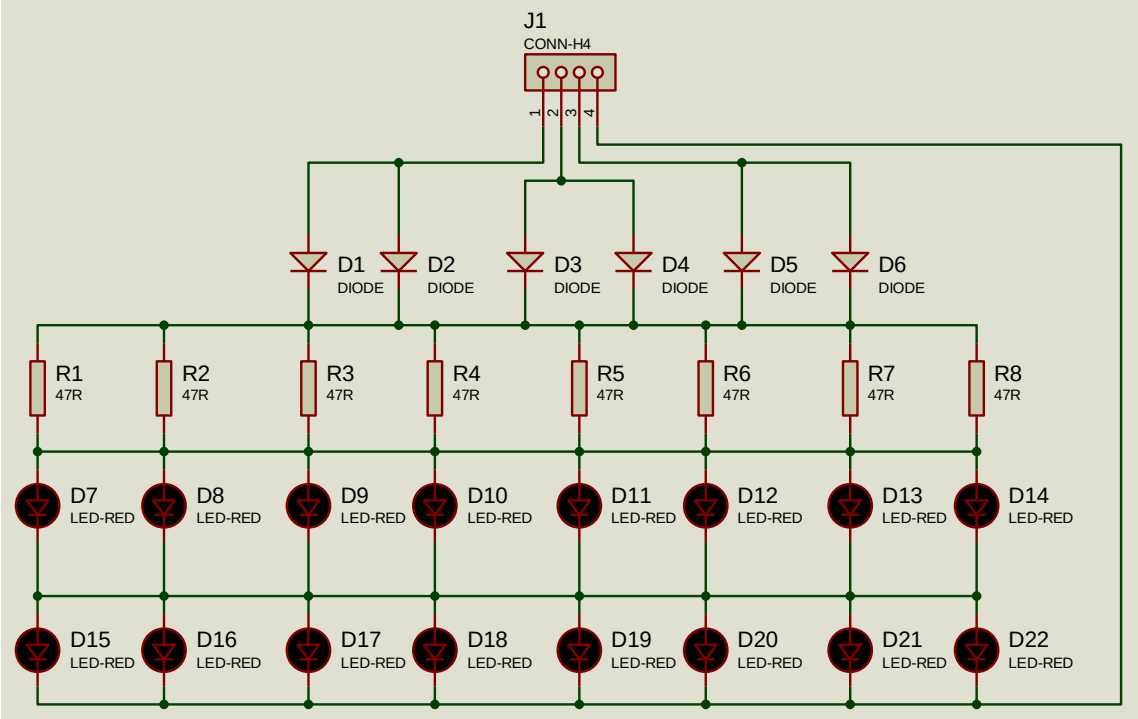
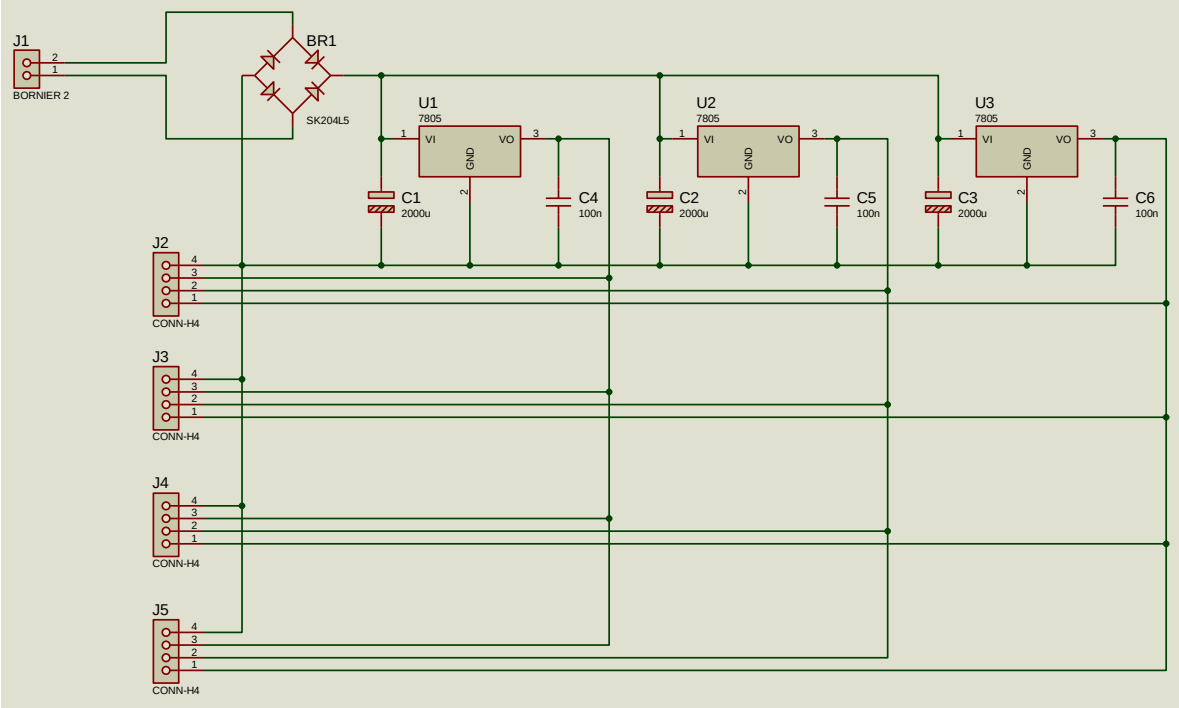


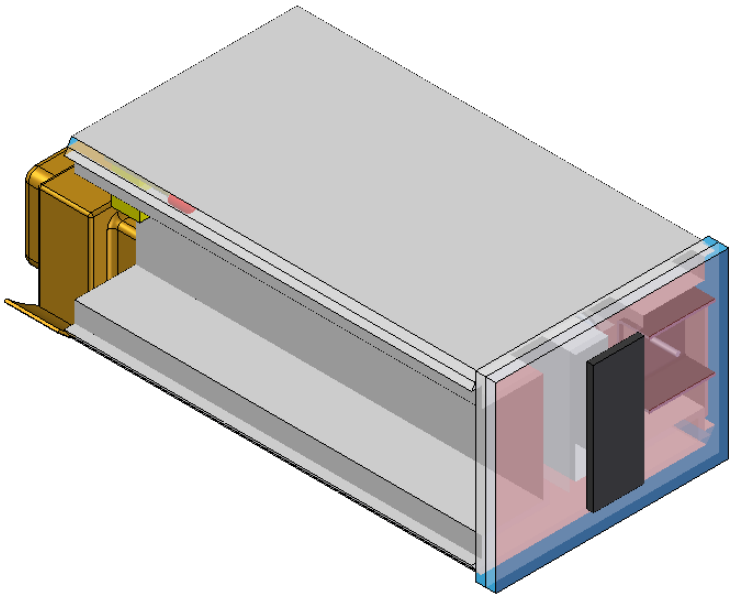
Ilustración 17. Esquema electrónico de la fuente del módulo de luces



Las tarjetas de LEDS al igual que la fuente se encuentran contenidas en un cilindro y soportadas por un juego de placas reflectoras que sostienen los LEDs alineados y en posición formando la figura de interés. Las placas reflectoras están diseñadas para guiar el chorro de los LEDs sin producir sombras. Se coloca frente a los LEDs y las placas que los soportan una pantalla difusora construido con cristal esmerilado para volver homogénea la luz que producen.

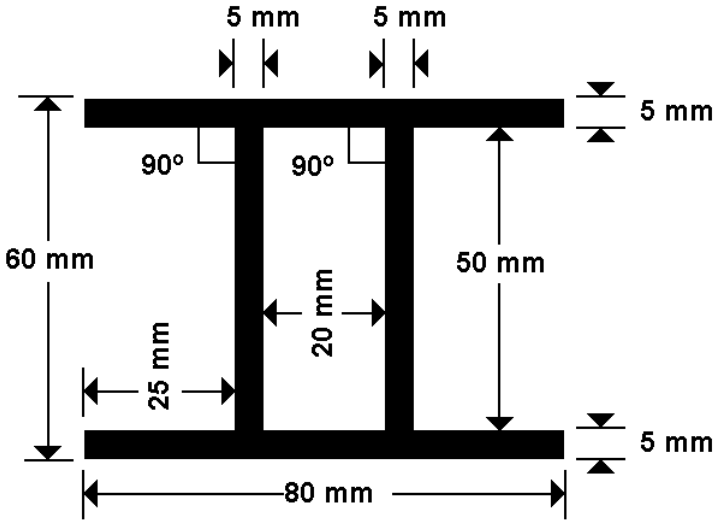
De esta manera se produce poca distorsión de la imagen con el ángulo de observación .Para homogeneizar el color se coloca frente a la pantalla difusora un vidrio rojo que actúa como filtro de color y evita la superposición de otras fuentes luminosas que puedan reflejar su luz sobre la figura hacia la cámara. Esta disposición se aprecia en la ilustración 18.

Ilustración 18. Estructura de placas y pantallas para formar internamente la figura



En frente de la pantalla difusora y el vidrio de color se coloca una rejilla que tiene la forma del objeto que se espera medir con el sistema de visión artificial, de tal manera que la luz que atraviesa la rejilla constituye el objeto luminoso o emisivo y la sombra que hace dicha rejilla pasa a ser el fondo de la imagen. Para el caso de estudio se ha escogido la figura de la ilustración 19 con las dimensiones mostradas:

Ilustración 19. Forma y dimensiones del objeto emisivo visto por la cámara



Esta forma ofrece al sistema de visión la suficiente información para determinar su posición dentro de la imagen y su conversión a unidades físicas, así como la calibración del sistema basado en características como:

- La figura tiene dimensiones externas de 80 mm de ancho por 60 mm de alto
- Las líneas paralelas tienen una longitud de 80 mm
- Las dos líneas horizontales son paralelas y están separadas 50 mm entre ellas
- Las dos líneas verticales son paralelas y están separadas 20 mm entre ellas.
- Las líneas verticales forman ángulo recto con las horizontales
- Todas las líneas son enteramente rectas y de grosor constante de 5 mm
- Las intersecciones entre los ejes de las líneas y las esquinas formadas alrededor de dichas intersecciones son constantes respecto a la posición de la figura en la imagen
- La diagonal del recuadro interno es una constante y vale $\sqrt{20^2 + 50^2}\,mm \cong 53{,}8mm$
- La figura es simétrica con respecto al centro y susceptible de orientación.

A partir de las intersecciones, esquinas, posición del recuadro que contiene la imagen y el centro de masa de la figura se puede determinar la posición de esta en la imagen. El centro de masa de la figura, por su simetría, se encuentra en el centro del recuadro que la contiene y se puede hallar en la intersección de las diagonales del rectángulo interno o con el promedio de las posiciones de las intersecciones.

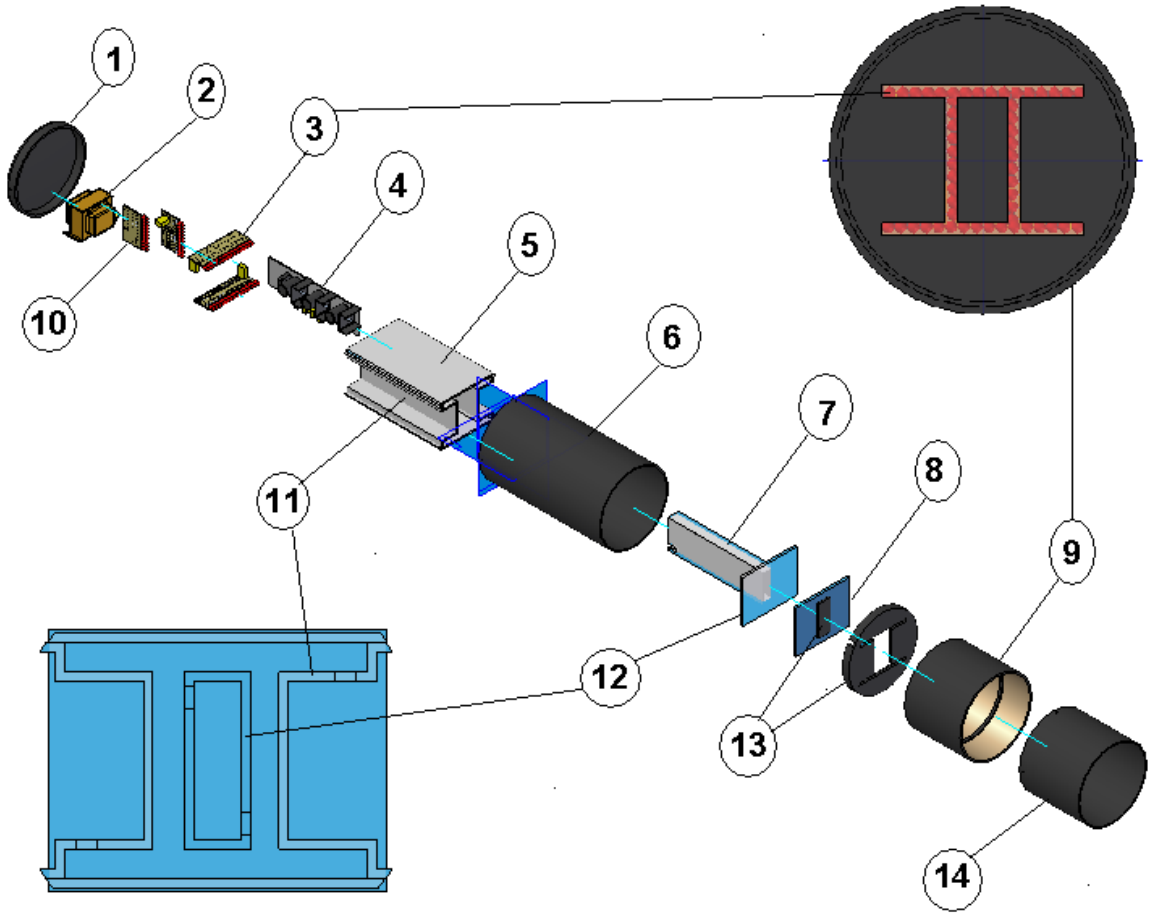
A partir de las dimensiones de la figura en píxeles en cuanto a distancias, longitudes y tamaños de la imagen previamente interpretados como dimensiones físicas se puede calibrar el sistema para traducir la posición del objeto en píxeles a unidades físicas de ingeniería.

Al observar la pendiente de las líneas horizontales y/o verticales se puede determinar el giro de la figura en la imagen en grados o radianes en un rango (-90º, 90º).

Debido al paralelismo y la figura basada en rectas se puede determinar los fenómenos ópticos de aberración de cojín y aberración de barril debido a las lentes. Una vez determinado esto se puede iniciar un proceso de reconstrucción de la imagen.

En la ilustración 20.se muestra todos los componentes que constituyen el sistema de luces como son: (1) Tapa posterior, (2) Transformador reductor de voltaje, (3) tarjetas de 16 LEDs (8 cm), (4) fuente de alimentación DC, (5) placas reflectivas horizontales, (6) cilindro de cubierta, (7), Placas reflectivas internas, (8) vidrio de color rojo, (9) unión, (10) tarjetas de 10 LEDs (5) cm, (11) placas reflectivas verticales, (12) vidrio difusor esmerilado, (13) rejilla formadora de la figura y (14) ceja protectora.

Ilustración 20. Vista explosionada ampliada del módulo de luces de referencia



A.3.1. Codificación del algoritmo de procesamiento de imágenes

El algoritmo diseñado finalmente es codificado en lenguaje ANSI C para que opere sobre un DSP Texas Instruments TMS320C6416-T, el cual es el más potente del mercado. A continuación se lista el código fuente del programa principal y las funciones que lo componen.

Programa principal

```
//desplazamiento.c
```

Creado el 13 de mayo de 2007
Modificado el 24 de octubre de 2007

Descripción:

El programa desplazamiento obtiene una imagen desde el PC, la almacena en una localidad de memoria y le aplica un algoritmo de procesamiento basado en los modelos SRI para determinar la posición de un objeto conocido.

El objetivo de este programa es determinar la carga computacional de cada una de las funciones y pasos de procesamiento del algoritmo.

```
/*******************************************************
*     Archivos de cabecera                             *
*******************************************************/
#include <stdio.h>
#include <stdlib.h>
//#include <math.h>
#include <rtdx.h>
#include "target.h"

RTDX_CreateInputChannel(ichan);
/*******************************************************
*     Definición de constantes                         *
*******************************************************/
#define     ALTO  1024        //Número de filas
#define ANCHO     1280        //Número de columnas
#define PIXELES   1310720     //Número de píxeles
#define TAMANO    40960
```

```
                              //Tamaño  de imagen binaria en palabras de
                              //32 bits
#define T_CODE    694000
                              //Tamaño inicial en bytes del código RLE.
                              //Tiene en cuenta la RAM disponible
#define COLS      160
/*******************************************************
*     Declaración de variables globales                        *
*******************************************************/
//Imagen que contiene el plano rojo procesado por el filtro
unsigned char I[ALTO][ANCHO];
int buf[PIXELES];
int X0, Y0;                                  //Posición cero inicial o punto
de referencia
float x, y;

/*******************************************************
*     Declaración de funciones                                  *
*******************************************************/

// Principales
int almacenamiento(int *in, int tam);
char adquirirImagen(int *in, unsigned char (*O)[ANCHO], unsigned int *h,
int tam);
char medirDistancias(unsigned short (*p)[2], int *kh, int *kv);
char calcularDesplazamiento(float c[3][2], char nObjetos, char nLineas,
      float *x, float *y, int *kh, int *kv);

void filtroPromedio(void);
void enviarSalida(void);

/*******************************************************
*     PROGRAMA PRINCIPAL                                    *
*******************************************************/
void main(void) {

      int tam;                        //Dimensión del vector de código RLE

      unsigned short roi[4];  //Posición de la ROI [xmin, ymin, h, w]
      float centro[3][2];             //Colección de puntos centros del
objeto
      unsigned int *h;  //Histograma de la imagen
      int *Ib;          //Imagen binaria. Cada bit representa un píxel
      int *Ib1;         //Imagen binaria auxiliar
      signed short *c;  //Código RLE para la imagen binaria
      unsigned char *e;       //Vector de etiquetas asociado al código RLE
      unsigned char nOb;      //Número de objetos detectados en la imagen
      int i;                  //Índice para control del proceso
      short lineas[4][2];     //Matriz de parámetros ro y theta de líneas
      unsigned short p[4][2];       //Puntos de intersección de los ejes
      int kh, kv;                   //Constantes de calibración.

      puts("\nINICIO DEL PROGRAMA...");
```

```
/***  ADQUISICIÓN DE LA IMAGEN      ***/

h = (unsigned int *) malloc(256 * sizeof(unsigned int));
if(adquirirImagen(buf, I, h, PIXELES)) {
      puts("Imagen capturada");
}
/***  SEGMENTACIÓN POR UMBRALIZACIÓN      ***/
Ib = (int *) malloc(TAMANO * sizeof(int));
if(umbralizacionAutomatica(I, Ib, h, 50, 120, TAMANO)){
      puts("Imagen umbralizada");
}
if(generarImagenDespliegue(Ib, Idisp, TAMANO)){
      puts("Imagen binaria Desplegada");
}
/***  ACONDICIONAMIENTO BINARIO     ***/
/***  Filtro morfológico      ***/
Ib1 = (int *) malloc(TAMANO * sizeof(int));
if(dilatacionBinaria(Ib, Ib1, ANCHO/32, TAMANO)){
      dilatacionBinaria(Ib1, Ib, ANCHO/32, TAMANO);
      dilatacionBinaria(Ib, Ib1, ANCHO/32, TAMANO);
      puts("Imagen binaria Dilatada");

      erosionBinaria(Ib1, Ib, ANCHO/32, TAMANO);
      erosionBinaria(Ib, Ib1, ANCHO/32, TAMANO);
      erosionBinaria(Ib1, Ib, ANCHO/32, TAMANO);
      erosionBinaria(Ib, Ib1, ANCHO/32, TAMANO);
      puts("Imagen binaria Erosionada");
}
if(generarImagenDespliegue(Ib1, I, TAMANO)){
      puts("Imagen Filtrada Morfológicamente Desplegada");
}


/***  Código RLE        ***/
free(Ib);
c = (signed short *) malloc(T_CODE);
tam =  codigoRLE(Ib1, c, ANCHO/32, TAMANO, T_CODE/2);
if(tam > 0){
      puts("Imagen codificada");
      nOb = 0;
      if(descomprimirRLE(c, Idisp, tam, ALTO)){
            puts("Imagen descomprimida");
      }
      free(Ib1);
      c = (signed short *) realloc(c, tam * sizeof(signed short));
      if(c){
            puts("Tamaño del codigo redimensionado");
           e = (unsigned char *) malloc(tam*sizeof(unsigned char));
            nOb = etiquetarCodigoRLE(c, e, tam);
            if(nOb){
                  puts("Código RLE etiquetado");
                  if(descomprimirEtiquetas(c, e, Irgb, tam, ALTO)){
                        puts("imagen de etiquetas generada");
                  }
```

```
                }

                /***  Filtro de Área    ***/
                if(nOb > 1){
                      nOb = removerObjetosPequeniosRLE(c, e, &tam, 25);
        //Remueve objetos de hasta 25%
                      if(nOb){
                            puts("Objetos pequeños eliminados");
                            descomprimirRLE(c, Idisp, tam, ALTO);
                            puts("Imagen descomprimida");
                      }
                }

                /***  EXTRACCIÓN DE CARACTERÍSTICAS ***/
                /***  Centroide         ***/
                if(medirCentroide(c, centro, tam)) {
                      puts("Centroide obtenido");
                }
          }// Fin de if(c);

          /***  Región de Interés ROI               ***/
          if(obtenerRoiRLE(c, roi, tam)){
                puts("ROI definida");
                centro[1][0] = roi[0] + roi[2]/2;
                centro[1][1] = roi[1] + roi[3]/2;
          }

          /*** TRANSFORMADA DE HOUGH                        ***/
          /*** Erosiones múltiples sobre el código RLE ***/
          for(i = 0; i < 5; i++){
                tam = erosionarCodigoRLE(c, tam);
          }
          if(tam){
                puts("Imagen erosionada mediante el código");
                descomprimirRLE(c, I, tam, ALTO);
          }

          /*** Determinación de lineas ejes         ***/
          i = detectarEjesRLE(c, lineas, tam, ANCHO);
          if(i > 3) {
                puts("Lineas Eje detectadas");
                i = obtenerIntersecciones(lineas, p, i);
                if(i == 4){
                     centro[2][0] = (p[0][0]+p[1][0]+p[2][0]+p[3][0])/4;
                     centro[2][1] = (p[0][1]+p[1][1]+p[2][1]+p[3][1])/4;
                }
                if(medirDistancias(p, &kh, &kv)){
                      puts("Sistema calibrado");
                }
          }
          calcularDesplazamiento(centro, nOb, i, &x, &y, &kh, &kv);
      }// Fin de if(tam > 0);
      puts("... PROGRAMA TERMINADO");
}
```

Función de almacenamiento

Función para recibir un flujo de datos de entrada y almacenarlos en una variable del programa, al leer un canal RTDX.

```
/***  ADQUISICIÓN DE LA IMAGEN      ***/
int almacenamiento(int *in, int tam){
      int s;
      //Evaluación de posibles fallas
      if(in==0){
            return 0;
      }

      TARGET_INITIALIZE(); //Contenida en target.h

      //Habilitar canal de entrada de datos
      RTDX_enableInput( &ichan );

      //Recibir el arreglo de enteros del Host
      s = RTDX_read( &ichan, &in, tam );
      if(s != tam){
            s = 0;
      }

      //Deshabilitar el canal antes de salir
      RTDX_disableInput(&ichan);
      return s;
}
```

Función de adquisición de imagen

Permite obtener la imagen filtrada por color con la que se inicia el proceso de medición de traslación.

```
char adquirirImagen(int *in, unsigned char (*O)[ANCHO], unsigned int *h,
int tam){

      int r, g, b;      //Variables para almacenar un píxel
      unsigned char p;        //Valor del píxel filtrado
      int i, j, k;      //Índices de los arreglos

      //Evaluación de posibles conflictos
      if((in == 0) || (O == 0)){
            return 0;
```

```
	}
	if(h == 0){
		return 0;
	}

	for(i=0; i<256; i++){
		h[i] = 0; //Clarear el vector de histograma
	}

	//Recorrer el búfer de entrada y aplicar filtro de color
	k = 0;
	for(i = 0; i < ALTO; i++){
		for(j = 0; j < ANCHO; j++){
			r = ((in[k] >> 16) & 0xFF);   //Extraer Byte 2
			g = ((in[k] >> 8) & 0xFF);    // Extraer Byte 1
			b = (in[k] & 0xFF);                 //Extraer Byte 0

			r = (2 * r) - g - b; //Filtro de color
			if (r < 0){
				p = 0;
			} else if(r > 255){
				p = 255;
			} else p = r;
			h[p]++;
			O[i][j] = p;
			k++;
			if(k == tam){
				return 1;
			}
		}
	}
	return 1;
}
```

Función para medir distancias

Esta función toma los cuatro puntos que se hallan con los ejes de la figura y después de determinar su posición en ella, se miden las distancias para determinar las constantes de calibración que permiten entregar las medidas de traslación en unidades físicas.

```
char medirDistancias(unsigned short (*p)[2], int *kh, int *kv){
	char i;
	unsigned short r[4][2];
	int d, dmin, dh0, dh1, dv0, dv1;

	//Evaluación de posibles conflictos
	if(((p == 0) || (kh == 0)) || (kv == 0)){
		return 0;
	}
	//Ordenar los puntos para establecer distancias
	dmin = rms(ALTO,  ANCHO);
```

```
for(i = 0; i< 4; i++){
      d = rms(p[i][0], p[i][1]);    //Distancia
      if(d<dmin){
            r[0][0] = p[i][0];
            r[0][1] = p[i][1];
            dmin = d;
      }
}

dmin = ALTO;
for(i = 0; i< 4; i++){
      d = p[i][0] - r[0][0];
      if(d < 0){
            d = -1 * d;
      }
      if((d < dmin) && (p[i][1] != r[0][1])){
            r[1][0] = p[i][0];
            r[1][1] = p[i][1];
            dmin = d;
      }
}

dmin = ANCHO;
for(i = 0; i< 4; i++){
      d = p[i][1] - r[0][1];
      if(d < 0){
            d = -1 * d;
      }
      if((d < dmin) && (p[i][0] != r[1][0])){
            r[2][0] = p[i][0];
            r[2][1] = p[i][1];
            dmin = d;
      }
}

d = 0;
for(i = 0; i< 4; i++){
      dmin = rms(p[i][0], p[i][1]);
      if(d < dmin){
            r[3][0] = p[i][0];
            r[3][1] = p[i][1];
            d = dmin;
      }
}

//Calcular distancias
dh0 = rms((r[1][0] - r[0][0]), (r[1][1] - r[0][1]));
dh1 = rms((r[3][0] - r[2][0]), (r[3][1] - r[2][1]));
dv0 = rms((r[2][0] - r[0][0]), (r[2][1] - r[0][1]));
dv1 = rms((r[3][0] - r[1][0]), (r[3][1] - r[1][1]));

//Calcular constantes
*kh = 50000 / (dh0 + dh1);    //Distancia horizontal real: 25000 um
```

```
	*kv = 110000 / (dv0 + dv1); //Distancia vertical real: 55000 um

	*kh = 13234 * (*kh) /10000;   //Compensación de las constantes
	*kv = 13234 * (*kv) /10000;
	return 1;
}
```

Función para calcular el desplazamiento del objeto

Esta es una función de interpretación de resultados que determina cual de las medidas de posición del objeto tiene mayor peso en virtud del número de objetos y ejes encontrados.

```
char calcularDesplazamiento(float c[3][2], char nObjetos, char nLineas,
                                          float *x, float *y, int *kh, int
*kv){

	//Evaluación de posibles conflictos
	if(c == 0){
		return 0;
	}

	//Condiciones de ponderación de las medidas
	if (nObjetos ==1){
		if(nLineas > 3){
			X0 = (7 * c[0][0] + 2 * c[2][0] + c[1][0]) / 10;
			Y0 = (7 * c[0][1] + 2 * c[2][1] + c[1][1]) / 10;
		} else{
			X0 = (7 * c[0][0] + 3 * c[2][0]) / 10;
			Y0 = (7 * c[0][1] + 3 * c[2][1]) / 10;
		}
	}else {
		if(nLineas > 3){
			X0 = (2 * c[0][0] + c[2][0] + 7 * c[1][0]) / 10;
			Y0 = (2 * c[0][1] + c[2][1] + 7 * c[1][1]) / 10;
		} else {
			X0 = (3 * c[0][0] + 7 * c[2][0]) / 10;
			Y0 = (3 * c[0][1] + 7 * c[2][1]) / 10;
		}
	}
	*x =  (*kv) * X0;
	*y =  (*kh) * Y0;
	return 1;
}
```

El resto de las funciones que hace uso el sistema se encuentran contenidas en las librerías *binario.lib* y *rle.lib*, creadas para dar soporte al algoritmo propuesto en este proyecto

En la librería *binario.lib* define la constante:

```
#define     ANCHO 1280
```

Y se encuentran las funciones:

```
char umbralizacionAutomatica(unsigned char (*restrict I)[ANCHO],
                             int *restrict O,
                             unsigned int *restrict h,
                             char m, unsigned char M,
                             int tam);

char umbralizacion(unsigned char (*restrict I)[ANCHO],
                   int *restrict O,
                   unsigned char t,
                   int tam);

char erosionBinaria(int *restrict I, int *restrict O, int c, int tam);

char dilatacionBinaria(int *restrict I, int *restrict O, int c, int tam);

char generarImagenDespliegue(int *restrict I, unsigned char (*restrict O)[ANCHO], int tam);
```

En la librería rle.lib se define la constante:

```
#define     ANCHO 1280
```

Y se encuentran las funciones:

```
int codigoRLE(int *Ib, signed short *c, int cols, int tam, int tc);

unsigned char etiquetarCodigoRLE(signed short *c, unsigned char *e, int t);
```

```
char removerObjetosPequeniosRLE(signed short *c,
                                unsigned char *e,
                                int *t,
                                unsigned int p);

char medirCentroide(signed short *c, float (*centro)[2], int t);

char obtenerRoiRLE(signed short *c, unsigned short *roi, int t);

int erosionarCodigoRLE(signed short *c, int t);

char detectarEjesRLE(signed short *c, short (*lineas)[2], int t, int rmax);

char obtenerIntersecciones(short (*lineas)[2], unsigned short (*p)[2], char n);

char descomprimirRLE(signed short *c, unsigned char  (*O)[ANCHO], int t, int alto);

char descomprimirEtiquetas(signed short *c,
                           unsigned char *e,
                           unsigned char (*I)[ANCHO][3],
                           int t,
                           int alto);
```

Funciones auxiliares para cómputo matemático

```
int rms(short x, short y);
int coseno(int angulo, int componente);
int seno(int angulo, int componente);
```