UNIVERSIDAD DON BOSCO.
FACULTAD DE INGENIERIA.
ESCUELA DE ELECTRONICA.

"Aplicación del Reconocimiento Optico de Objetos en un Brazo Robot dentro de un Entorno Controlado"

PROYECTO DE GRADUACION PARA OPTAR AL GRADO DE INGENIERO, OPCION EN ELECTRONICA.

UNIVERSIDAD DON BOSCO Biblioteca Central

PRESENTADO POR:

EDWIN ANTONIO GAMERO BONILLA **94-01158**

JOSE ORLANDO ALAS RODRIGUEZ **94-01020**

ASESOR DEL PROYECTO:

ING. OSCAR DURAN VIZCARRA.

CIUDADELA DON BOSCO, SEPTIEMBRE 20, 1999

Ing. Oscar Durán Vizcarra.
ASESOR.

Ing. Federico Laínez.
JURADO.

Ing. Mauricio Flores.
JURADO.

Dedicatoria y Agradecimientos de Orlando Alas.

Todo este trabajo se lo quiero dedicar a mi madre Antonia Margarita Rodriguez de Duarte quien ha esperado y luchado mucho por que este momento de triunfo llegue. Además quiero agradecerle a muchas personas que de una manera desinteresada han colaborado durante todo el camino que conllevó el presente trabajo. Me encantaría poder escribirles a cada uno de ellos palabras de agradecimiento pero lastimosamente no puedo, pero tengan por seguro que mi agradecimiento es de corazón. A continuación menciono todos los que colaboraron para realizar este trabajo:

Antonia Margarita Rodríguez de Duarte
Milagro Monge Sandoval.
Antonio Zura Peraza
Arturo Saravia
Roxana Flores
Juan Carlos Rivas
Remon Spekreijse
Charles Dijak
Kathy Abigail Alas
Leslie Ximena Duarte
Familia Gamero Bonilla
Pablo Gallardo
Francisca Sanchez
Fabiola, Helena y Paola Rodriguez Sánchez
René Mauricio Gochez Chicas
Miguel Angel Lemus
Mauricio Tobar
Juanita Gallardo
Mario Jovel
Victor Cuenca
José Carlos Hernández
Jenny Sánchez.

A todos les agradezco muchísimo.

Dedicatorias y Agradecimientos de Edwin Gamero.

Deseo ofrecer este logro de mi vida primeramente a Dios que es el origen de mi motivación para superarme y hacer todo lo mejor que mis fuerzas me permitan. Gracias Señor por el regalo de la vida y por todas las bendiciones que me has dado durante todos estos años. Te he podido descubrir en todos los acontecimientos de mi existencia y ese es el más grande de los regalos, gracias. Mi futuro profesional lo dedico a Ti.

Doy gracias a mis padres por la ayuda que siempre me han dado, por sus consejos y por todo el tiempo que dedicaron en formarme. Esa es la mejor herencia que podían darme. Este triunfo lo dedico a ustedes.

Agradezco a mi hermano Walter la paciencia que me ha tenido en tantas ocasiones que todo ha sido solamente trabajo. Este logro es también tuyo y para ti.

Quiero aprovechar la ocasión para agradecer al Padre Pedro García por haberme dado, más que consejos, su amistad. Gracias por confiar en mi, y deseo expresarle que usted ha sido un ejemplo en mi vida por el carisma de autenticidad que le caracteriza. Gracias por su apoyo y confianza.

Doy gracias a la familia de Orlando que tantas veces les hicimos desvelarse con el trabajo de esta tesis. Gracias por su hospitalidad, amabilidad y amistad. De manera especial agradezco al tio de Orlando, Don Pablo Gallardo, porque me recibió en su casa como si fuera su sobrino cuando desarrollabamos este proyecto en su etapas finales.

Mil gracias a todos mis amigos, que serían muchos para nombrarlos uno a uno, que siempre creyeron en mi y me dieron ánimos para continuar en los momentos de fatiga. Gracias a todos por su amistad y sepan que cuentan conmigo.

Indice de Contenido.

CAPITULO III.
FUNDAMENTACION EXPERIMENTAL PARA EL DESARROLLO DEL PROYECTO.

CAPITULO IV.
ALGORITMOS DESARROLLADOS.

Indice de Figuras.

CAPITULO I

GENERALIDADES DEL PROYECTO.

1.1 JUSTIFICACION DEL PROYECTO.

El presente proyecto pretende crear un gran impacto académico y dar uno de los primeros pasos para la implementación de un sistema mecánico-computarizado de brazo robot con un control flexible y capaz de extraer su información de la situación misma.

Se habla mucho de que nuestro país tiene retraso tecnológico pero no se visualizan o no se quieren ver, los pasos necesarios para sacarle de esa situación. Uno de los pasos indispensables es desarrollar investigación propia, que se apegue a las realidades de El Salvador. El grupo de esta tesis desea hacer un aporte a nuevos conocimientos originados y aplicados en el país.

Un brazo robot que tenga flexibilidad en desarrollar sus tareas debido al uso de un reconocimiento de objetos tiene una amplia gama de aplicaciones:

- Control de calidad en la que se selecciona en base a la forma o los colores.
- Construcción industrial de piezas o equipos por medio de manufactura de formas.
- Manipulación de utensilios que se encuentran en zonas de alto riesgo para el humano.
- Exploración y recolección de muestras, tal es el caso de los robots móviles enviados a Marte y la Luna, o los utilizados en la industria minera.
- Manipulación semi - inteligente de objetos diversos y pesados.
- Soldadura realizada por robot en la que se evalúa la calidad a través de una imagen.
- Cirugía ejecutada por un robot supervisada por humano.
- Armado de vehículos, aviones y electrodomésticos por medio de brazos mecánicos con cámaras.
- Soldadura microscópicas en pastillas de circuitos integrados.

- Y la frontera de la robótica que es un sistema inteligente capaz de interactuar con su entorno.

El presente proyecto será una de las primeras investigaciones y aplicaciones (ambas al mismo tiempo) a nivel nacional sobre el procesamiento de imágenes aplicada a la automatización; sentándose una base para que futuros proyectos aporten soluciones más completas o desarrollen otras aplicaciones afines. Entre las futuras ideas para tesis que se pueden apoyar en esta línea se encuentran: sistemas que fabriquen piezas mecánicas en base a fotografías o dibujos, sistemas de control de calidad de vegetales y granos en base al color, un equipo que sea capaz de imprimir las diferentes vistas de una pieza y que dé características como su centro de masa, también el control automático en la tipografía y serigrafía (ajuste de los controles de impresión basados en muestras del producto final).

Las fuentes que se disponen de proyectos similares a éste han sido realizados en ciertas universidades extranjeras, especialmente europeas, en las que se desarrollaron o actualmente se están trabajando proyectos con brazos robot. Esta información se encuentra en Internet, pero no siempre se presenta de manera completa por razones obvias. Es necesario tener información e investigación propia.

1.2 OBJETIVOS.

1.2.1 OBJETIVO GENERAL.

Diseñar e implementar un sistema computarizado que permita captar imágenes, procesarlas, reconocer objetos de las mismas y manipular a éstos por medio de un brazo robot.

1.2.2 OBJETIVOS ESPECIFICOS.

- Desarrollar un software que permita el reconocimiento de objetos.
- Diseñar el algoritmo y programa que permitirá la manipulación de los objetos por medio del brazo robot en base a la imagen; ubicándose espacialmente las piezas en base a la misma.
- Implementar los medios mecánicos y eléctricos necesarios para el control del movimiento del brazo robot.
- Desarrollo de una interfaz de usuario de fácil utilización, y que se base en ambiente Windows 95 / 98 para la presentación y control del sistema de reconocimiento de objetos - brazo robot.

1.3 ALCANCES Y LIMITACIONES.

1.3.1 ALCANCES:

- El compromiso principal del proyecto es realizar un sistema que permita aplicar en un prototipo los métodos básicos del procesamiento de imágenes para realizar un reconocimiento de objetos.

- El software aplica funciones tales como detección de bordes, filtros, transformaciones morfológicas, cambio de contraste, umbralizado y aritmética entre imágenes.

- El sistema es fácil de usar por tener una interfaz gráfica (GUI).

- El principio básico de este sistema es aplicable a futuros proyectos.

- Se utiliza programación en plataformas gráficas de 32 bits (Visual C++ , versión 5.0 Profesional).

- El brazo realiza su posicionamiento únicamente por medio de las coordenadas suministradas por el procesamiento de la imagen.

- Se realizan algoritmos que permiten la ubicación espacial completa de los objetos a partir de la imagen.

- El reconocimiento de objetos se aplica a una pieza en movimiento que se desplaza a velocidad lenta, permitiéndo al brazo seguirla.

1.3.2 LIMITACIONES:

- Se trabajan imágenes formadas por diferentes tonos de grises (con 8 bits por pixel).

- El proyecto es un prototipo para investigación de ingeniería y no es una aplicación inmediata a la industria.

- El proyecto no pretende ser un analizador de imágenes (que contenga todos los algoritmos conocidos para el procesamiento de imágenes) sino que se desarrollan solamente los algoritmos necesarios para la aplicación.

- El sistema no se comporta adecuadamente con objetos que cubren totalmente a otros.

- El usuario no podrá alterar los parámetros de las imágenes sino que se limitará a seleccionar el objeto a mover.

- El brazo sólo se comportará adecuadamente en el entorno controlado, que tiene dimensiones, objetos, colores y forma predeterminadas.

- Los objetos a usar para las pruebas son de formas definidas (no polimorfas), con peso limitado, superficies no lisas, oscuros y de dimensiones definidas.

- No se pueden recordar figuras, es decir, que no se tiene memoria de las figuras previamente encontradas.

- La aplicación para el seguimiento de objetos se puede hacer únicamente con uno. No deben de haber dos o más objetos en la escena para este algoritmo.

CAPITULO II

BASES TEORICAS PARA EL DESARROLLO DEL PROYECTO.

2.1 LA CAMARA .

2.1.1 LA CAMARA POR DENTRO.

Cada fabricante utiliza su propia tecnología, por lo que los componentes dentro de la cámara cambian de apariencia notablemente, sin embargo, el uso que tienen es el mismo. En la figura de la parte inferior se muestra una cámara con sus detalles internos. De entre los elementos más importantes tenemos: el tubo de cámara y la lámina fotosensible.

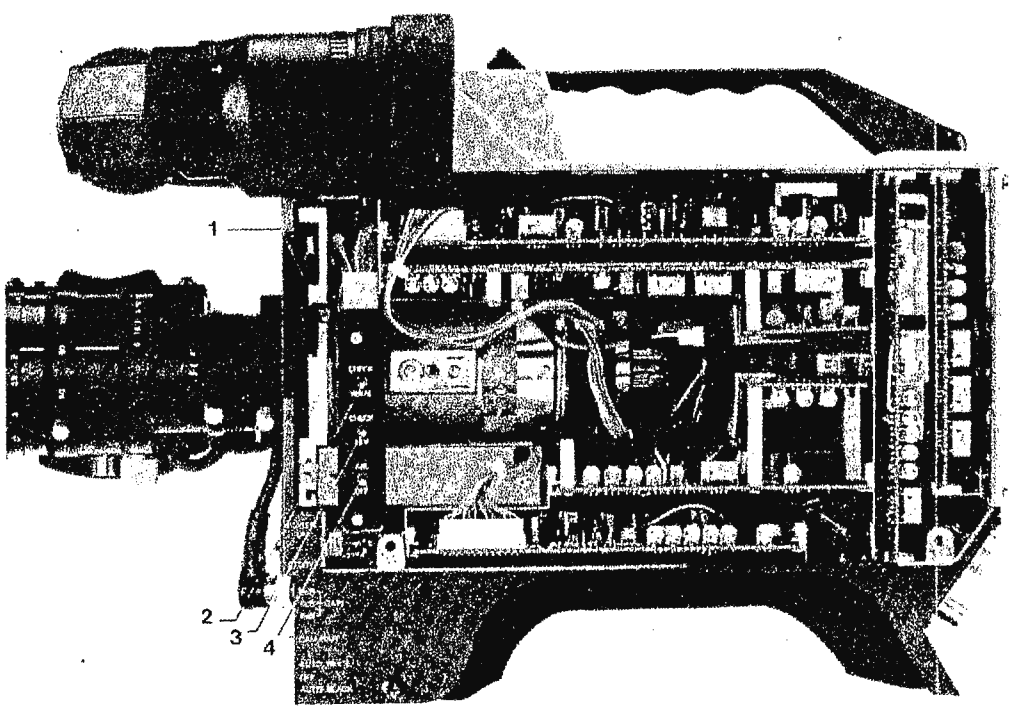


Fig. 1. Corte transversal de una cámara Vidicom.

EL TUBO DE LA CAMARA.

El tubo de cámara transforma la señal luminosa capturada a través del lente objetivo de la cámara en señal de video, la cual una vez amplificada, se envía a un sistema de grabación (magnético, dentro de una PC, etc.) o se transmite por algún medio.

El proceso para la transformación de la señal luminosa captada a través del objetivo de la cámara en señal de video, dentro del tubo es la siguiente:

- La imagen que capta el objetivo de la cámara (en realidad, diferentes intensidades de luz), se forma a su vez sobre la lámina fotosensible conocida como "mosaico", la cual altera sus características electrónicas por acción de la luz que incide sobre ella.
- Simultáneamente, un haz de electrones recorre y analiza dicha lámina, el cual explora la totalidad de la imagen por medio de un circuito de barrido horizontal y otro de barrido vertical, generándose una señal eléctrica de video que será la que llegue la etapa de almacenamiento o la de emisión.

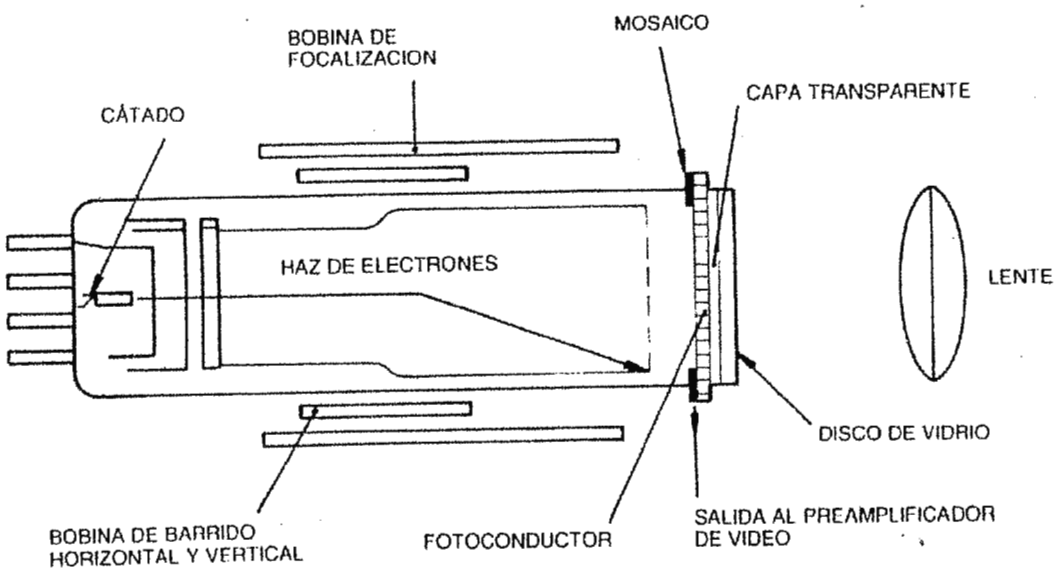


Fig. 2. Esquema simplificado de un tubo Vidicom

2.1.2 TIPOS DE CAMARA.

Cuando se elige una cámara deben de tenerse en cuenta muchas consideraciones y en general se llegará a una especie de compromiso entre las diversas cualidades. Sin embargo, si se establecen las prioridades en un orden correcto, seguramente se tomará la decisión adecuada.

- ***Formato:*** Un primer factor a tener en cuenta es el formato en el que trabaja la cámara: VHS u 8 mm, cada uno de ellos con sus versiones compuestas (VHS y 8) o por video separado (S-VHS y Hi8), de mucha mayor calidad pero también de mayor precio.
- ***Sensor:*** El objetivo de la cámara tiene la misión de enfocar la imagen que se desea captar en la zona sensible del sensor. Uno de los elementos que más han contribuido a la disminución del tamaño de las cámaras es la introducción de sensores en estado sólido: el MOS y el CCD. Estos sensores están formados por una gran cantidad de células fotosensibles cuyas dimensiones varían entre los 8 y los 12 micrómetros. Los sensores en estado sólido más utilizados son de 1/3 de pulgada, 1/2 pulgada, con unas zonas sensibles de 4.4 x 3.3 mm, 6.6 x 5 mm, y 8.8 x 6.6 mm, respectivamente.
- ***Estabilizadores de imagen:*** Algunas cámaras están dotadas de este tipo de mecanismos, en especial de EIS (estabilizador electrónico de imagen), que detecta y corrige vibraciones o movimientos bruscos de la cámara y que consta de una memoria que es comparada con la imagen captada por el sensor. Si ambas no coinciden, la memoria suministra señal al magnetoscopio. Con el EIS, las vibraciones o movimientos bruscos de las tomas cámara a mano que dan totalmente eliminadas.
- ***Sensibilidad:*** Las cámaras varían ampliamente en lo que respecta a la sensibilidad, que es de vital importancia para las tomas en interiores. Las cámaras más sensibles de uso casero operan con sólo 3 lux (la luz de una vela), aunque la presencia de ruido es bastante alta en estas condiciones. Para obtener imágenes en condiciones óptimas, debe operarse a partir de 1400 lux (interiores bien iluminados).
- ***Ergonomía:*** La consideración del peso no puede separarse de todo el conjunto de facilidad de manejo, su ligereza y facilidad de manipulación. Esta característica depende mucho del gusto personal.

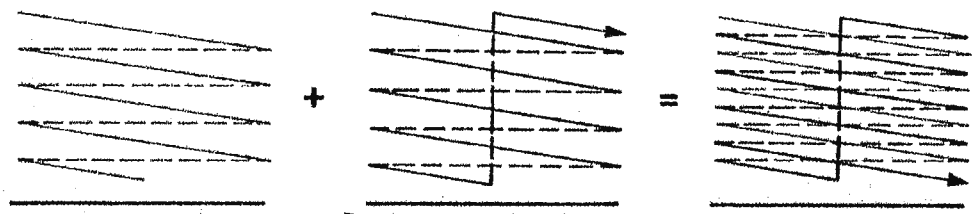
- ***Objetivos:*** Además del contraste y la definición, los principales factores a considerar cuando se elige un objetivo son: la máxima abertura y la relación de zoom. Supone una gran ventaja tener una lente de f1.4, pero advierta que con esta abertura la profundidad de campo es extremadamente pequeña. Tales lentes son más caras que las tradicionales de f1.8. También la relación de zoom se elige según la necesidad, los hay de 6:1 o de 10:1. Las hay también de autozoom y macro.

CAMARA TIPO VIDICOM

La forma más corriente de tubo para las cámaras domésticas o semiprofesionales, es el vidicom. La lente enfoca la imagen en una placa (de 17mm con 25mm de diámetro). La placa, denominada mosaico, está recubierta por la cara del lado de la lente con una película conductora pero transparente, de óxido de estaño y por la otra cara de una capa de material fotoconductor. La propiedad del material fotoconductor que hace posible la señal de video es que la resistencia eléctrica varía a través de la superficie según la cantidad de luz que incide en un punto dado. Todo lo que se necesita después es un método para examinar de forma organizada las distintas resistencias a través del mosaico, a fin de reconstruir la imagen en una pantalla de TV. Ello es posible mediante un rayo de electrones lanzando desde un cátodo en el extremo trasero del tubo: cuando un electrón alcanza el mosaico, su carga (modificada según la resistencia de aquel punto concreto, es decir, de acuerdo con la luz que cae en el punto) pasa a través de la superficie conductora exterior del mosaico como una pequeña parte de la señal de video. Para explorar todo el campo de la imagen, la fina corriente de electrones comienza con el ángulo de la parte superior izquierda de la placa y se mueve hacia la derecha, bajando ligeramente; cuando alcanza el borde de la derecha, el punto salta hacia atrás al borde izquierdo (un poco más abajo esta vez) para examinar la línea siguiente y así sucesivamente hasta la parte inferior de la placa. Cuando alcanza la parte inferior vuelve instantáneamente a la parte superior izquierda de la placa para explorar otro campo que estará entrelazado con el primero, para formar una imagen completa en 1/25 o en 1/30 de segundo. Este retorno rápido es conocido como "borrado" (fly back).

La dirección del rayo se controla por bobinas magnéticas de deflexión, que rodean el tubo y los movimientos del rayo están sincronizados por un impulso sincrónico horizontal y vertical; estos

son conocidos como "sincronismo de línea" y "sincronismo de campo", respectivamente. El sincronismo de línea indica al rayo cuando debe de volver a la izquierda a comenzar una nueva línea, mientras el sincronismo de campo contiene la instrucción de que vuelva a lo alto del mosaico para el próximo campo. El impulso vertical puede tomarse, bien sea de la corriente de alimentación (60Hz) o de un cristal oscilador existente en la propia cámara. El receptor de TV, a su vez, observa estos impulsos en la señal de video y reconstruye el proceso en la pantalla.



Campo 1. El primer proceso de exploración sólo examina la mitad del *campo*, como si se moviese a través de un césped sólo en la mitad, en bandas, en preparación para un segundo recorrido. Luego vuelve arriba para el segundo campo.

Campo 2. La segunda exploración o campo tiene en cuenta las bandas restantes alternadas en la siguiente fracción de segundo; 1/50 (Inglaterra) 1/60 (USA).

El cuadro completo. Juntos puede considerarse que han seguido todo el cesped; cubriendo todo el suelo en 1/25 (Inglaterra) o 1/30 (USA) de segundo. El cuadro completo compuesto de 2 campos está ahora terminado sin fallo visible.

Fig. 3. Trazos para formar la imagen de video en pantalla.

CAMARA TIPO CCD.

Están incluidas en la denominación genérica de Camcoder, que se ha popularizado por ser una generación de cámaras que poseen la ventaja de incorporar la cinta de grabación en el cuerpo de la cámara. Todas las CCD son de ese tipo, aunque ya no necesitan de una cinta, pues digitalizan de manera directa.

La razón por la cual ha sido posible equipar a la cámara con un equipo grabador integrado a la misma, se debe a la supresión del tubo de la imagen que incorporan las cámaras convencionales, el cual ha sido sustituido por un dispositivo de imagen del tipo CCD (Charge Coupled Device) de escasas dimensiones.

Al no existir el tubo, ni el mosaico es posible, por lo tanto, enfocar directamente sobre las grandes luces, incluso el sol, sin peligro de deterioro de los componentes electrónicos.

Los dispositivos CCD están constituidos por un pequeño sensor de imagen formado por un mosaico de silicio en el que se encuentran un gran número de fotodiodos distribuidos de manera

perfectamente homogénea; al incidir la luz sobre ellos se produce una corriente eléctrica a alta velocidad que genera la señal de video. Ver las figuras siguientes:

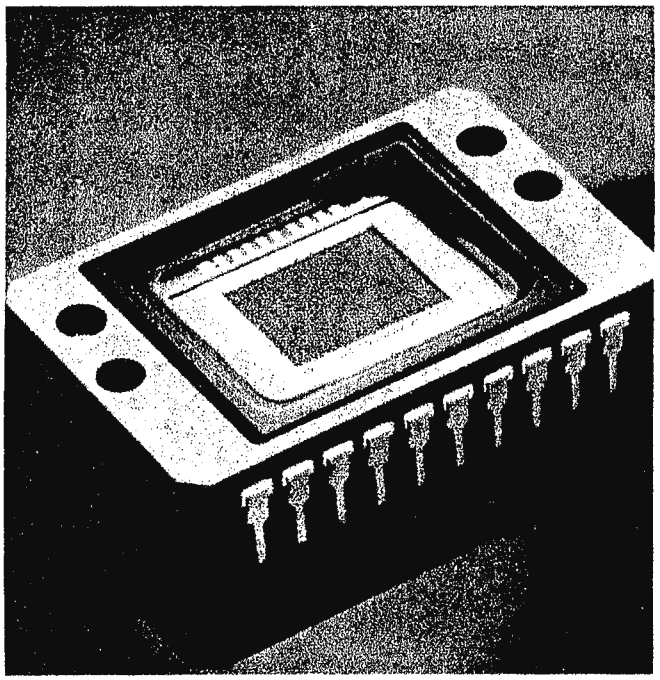
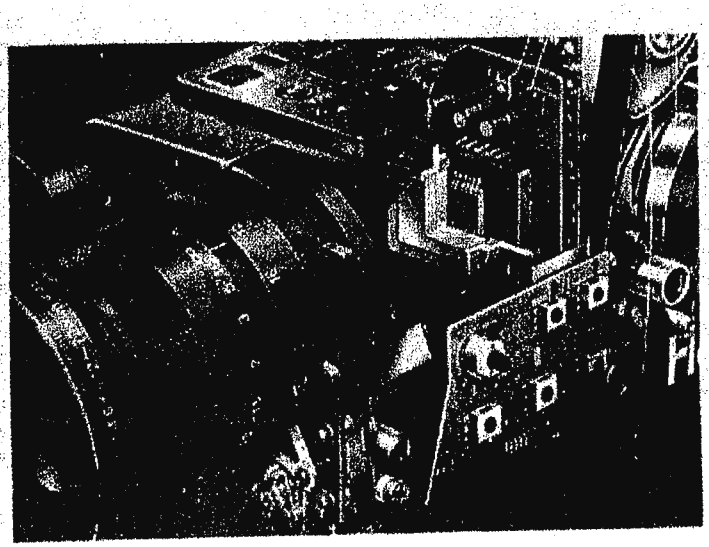


Fig. 4 Imagen ampliada de un chip CCD.



CCD. Las cámaras actuales, en las que la cinta de video se coloca en el interior del cuerpo de la cámara, utilizan un dispositivo de carga (CCD) en lugar del tubo convencional. Funciona de una forma similar con la diferencia de que se funda en las propiedades del estado sólido; un chip muy pequeño es explorado según una matriz microscópica de líneas verticales y horizontales grabadas en su superficie. Este tipo de tecnología de estado sólido ha sustituido a los tubos convencionales casi por completo.

Fig. 5 Corte interno de una cámara CCD.

Al incidir la luz sobre los fotodiodos de que está formado el sensor de la imagen, se produce una corriente eléctrica, a alta velocidad, que genera la señal de video.

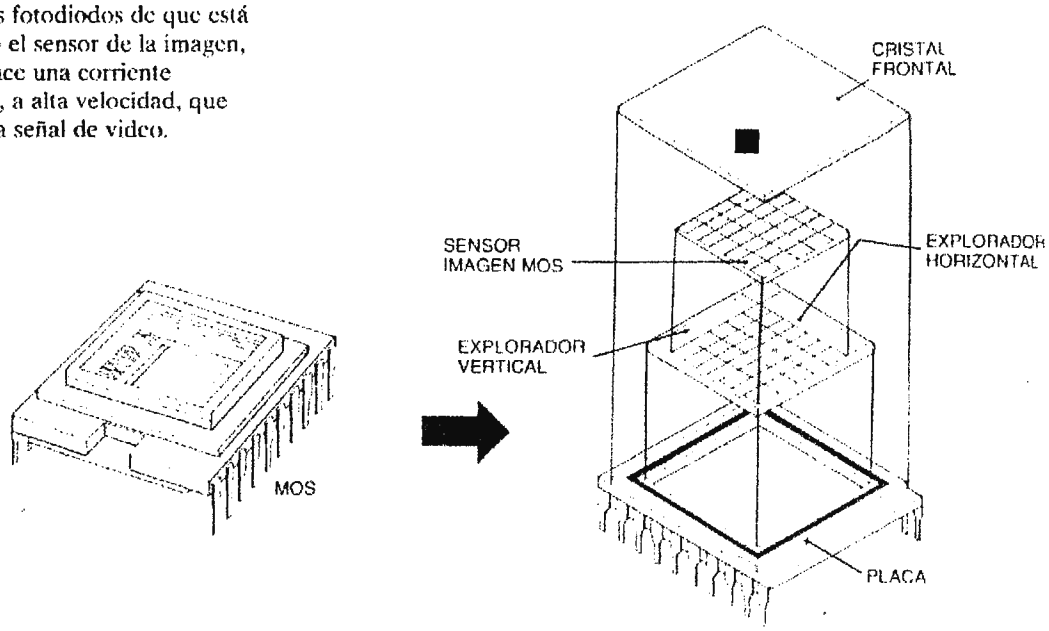


Fig 6 Corte interno de un chip de CCD.

2.1.3 DESCRIPCION DE LA RELACION ENTRE LA LOGITUD FOCAL Y EL ANGULO DE VISION.

En principio, todos los objetivos de idéntica longitud focal, poseen el mismo ángulo de visión, es decir, que abarcan el mismo campo.

Y decimos "en principio" porque puede encontrarse con la sorpresa de que dos objetivos que lleven la indicación de poseer la misma distancia focal, en la práctica existan diferencias entre ellos en cuanto al campo abarcado. Ello se debe a que una total exactitud en cuanto al calibrado de cada objetivo es algo muy difícil y sólo los grandes fabricantes consiguen tal precisión.

En la gráfica que se muestra en la siguiente página se expresa la relación entre las distintas longitudes focales y el ángulo de visión.

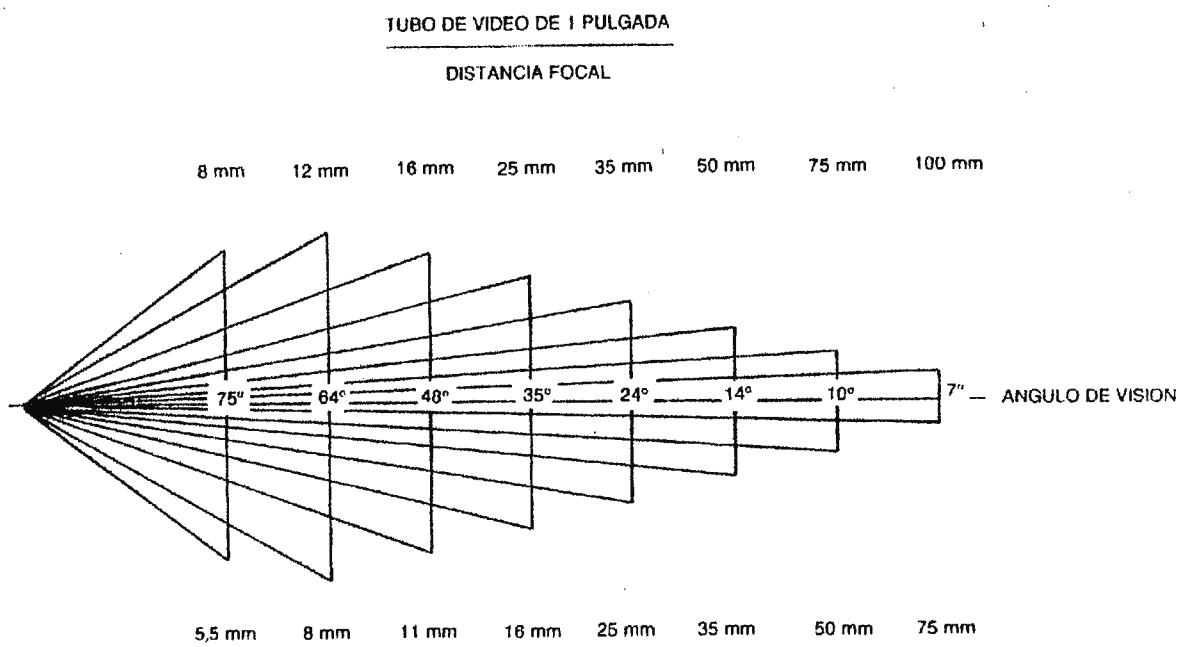


Fig. 7 Gráfica de relación entre ángulo de visión y distancia focal.

En cualquier cámara fija o móvil, el objetivo enfoca la luz que entra sobre un plano conocido como plano focal y en este punto es en el que está colocado el mosaico en el tubo de video. La imagen será de hecho invertida, pero esto se corrige electrónicamente. El ángulo de visión de los distintos objetivos varía según su distancia focal, que es la distancia desde el centro óptico o punto nodal del objetivo al plano focal. Cuanto mayor es la distancia focal, más cerrado es el ángulo visión: un teleobjetivo tiene una distancia focal grande y un ángulo de visión pequeño; un gran angular, con una distancia focal corta, proporciona un amplio ángulo visual. Esto tiene efectos muy importantes, no sólo en el tamaño de la imagen, sino también sobre la perspectiva que se obtiene.

Es importante recordar que el ángulo de visión proporcionado por la distancia focal de una lente, depende por completo del tamaño del mosaico del tubo. Cuanto mayor sea el mosaico más abierto será el ángulo de visión para una distancia focal dada.

2.1.4 ILUMINACION.

Por lo general, la calidad de la imagen de video es afectada por el nivel de iluminación de una manera drástica. En video la cantidad de luz se mide, bien sea en candelas-pie o en lux (lumen por metro cuadrado). Una candela-pie es igual a 10.76 lux. Una candela pie sería el nivel imaginario de iluminación que habría sobre una superficie que estuviese a un pie de distancia de un foco, que emitiese un lumen de luz; el lux es la equivalencia métrica. Debido a la ley del inverso de los cuadrados, al duplicarse la distancia de una fuente luminosa producirá una iluminación 4 veces menor sobre un objetivo determinado. En la práctica esto significa que su capacidad de grabar está limitada por el nivel de iluminación, desde unos 3 lux (una vela) hasta 1400 lux (un espacio interior bien iluminado)

Ley de la inversa de los cuadrados. Debido a la llamada ley de la inversa de los cuadrados, al doblarse la distancia desde una luz al objeto, el nivel de iluminación se reduce a la de 1/4 parte (en lugar de a la mitad, como podría esperarse). Este efecto se minimiza utilizando reflectores puntuales, pero siempre debe tenerse en cuenta.

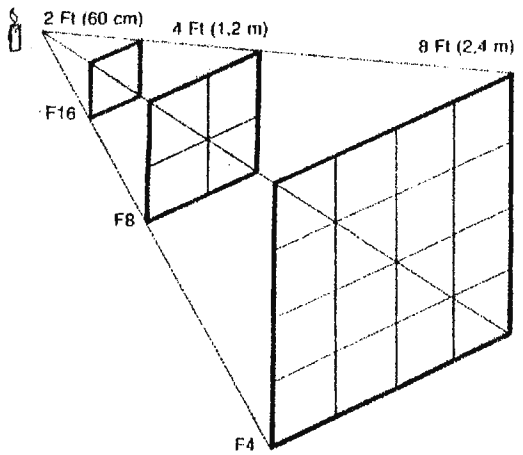
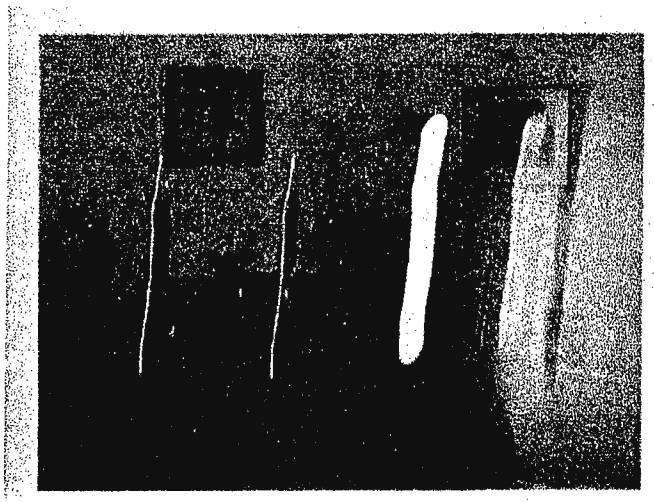


Fig. 8 Representación gráfica de la ley inversa de los cuadrados.

RETARDO (LAG) Y RUIDO.

La falta de exposición se manifiesta en el video en un aumento progresivo del nivel de "ruido" visual. En las cámaras no provistas de sensores sólidos, a medida que la sensibilidad del tubo se eleva automáticamente para hacer frente a las imágenes oscuras, el tubo se hace sensible al retardo (lag). El "lag' es el retraso producido cuando hay objetos brillantes en movimiento, en relación con el tubo, a bajo niveles de iluminación y es el defecto más serio del tubo de vidicom.

A este efecto se le conoce también como "cola de cometa". Los sistemas con CCD no presentan este efecto de corrimiento pero si una mayor sensibilidad a las variaciones en la iluminación.



Lag. Todos los tubos de cámara tienen el defecto denominado «lag» en mayor o menor grado, cuando se filman luces brillantes moviéndose en bajos niveles de iluminación. Las luces toman la forma de una cola de cometa extendiéndose detrás del foco de iluminación brillante. Los tubos modernos, tales como el Saticon, Plumbicon y Cosvicon, son menos susceptibles a estos defectos que los antiguos tubos vidicon. Con la aparición de los sensores en estado sólido, en especial el CCD, este problema ha quedado resuelto.

Fig. 9 Efecto de Retardo en una cámara.

2.1.5 GANANCIA Y ABERTURA.

La salida de la cámara está determinada por tres factores principales: el nivel de iluminación, el grado de ganancia de la cámara y la abertura del objetivo. En una cámara de video el control automático de la ganancia (AGC) aumentará dicha ganancia hasta que se haya alcanzado una salida satisfactoria y solamente entonces se disminuirá la abertura para reducir la luz que llega al tubo (en condiciones de sol brillante por ejemplo).

Esta es la razón por la que se encuentran problemas de ruido y retardo en condiciones de ganancia elevada. La abertura máxima viene determinada por el objetivo que ocupa la cámara o, en caso de objetivos intercambiables, por el que se utiliza. Cuando menor es el número focal, mayor es la abertura.

Ruido. El ruido de video que se produce a bajos niveles de iluminación, debido al consiguiente alto nivel necesario de ganancia de video, toma la forma de una coloración moteada especialmente en las áreas en sombra de densidad uniforme.

Fig. 10 Efecto del ruido en la toma de una imagen.

Es interesante señalar que cualquiera que sea la fuente luminosa (tungsteno, cuarzo o fluorescente), el diseño de un reflector tiene un efecto importante en la calidad de luz emitida: un reflector parabólico da un haz estrecho y concentrado de luz, un reflector difusor da un haz ancho y sin una misma dirección para cada componente lumínica.. Por otra parte los difusores concentrados, casi siempre pueden variarse para dar un haz duro y estrecho o un haz difuso (como en el diseño de Fresnel). Lo anterior se muestra en la siguiente figura:

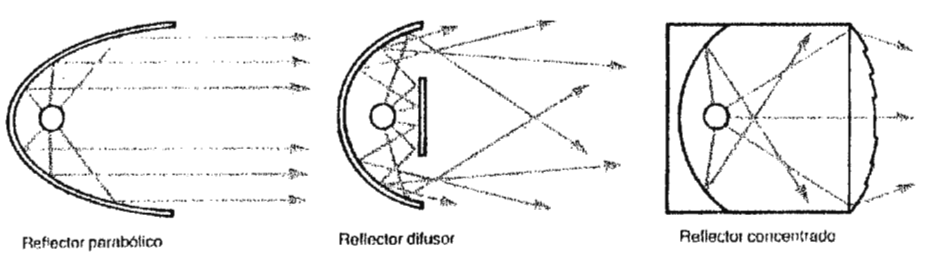


Fig.11 Diferentes tipos de reflectores que afectan a la iluminación.

2.1.6 CODIFICACION DEL COLOR Y SISTEMAS INTERNACINALES.

"Codificación" es el término utilizado para describir el proceso mediante el cual la información de color de la cámara (croma) se suma a la información de la luminancia, de tal manera que puede posteriormente, ser decodificada y mostrada por el receptor. Si a cada uno de los cuatro componentes (R,G, B, Y) se les atribuye la misma cantidad de espacio en la anchura de banda disponible, la anchura de bandea debería de ser enormemente grande; sin embrago el problema puede resolverse por una forma de "algebra" electrónica, que es la señal de luminancia Y. Esta señal se resta de la señal de rojo (R-Y) y de la azul (B-Y). Estas dos señales se modulan sobre una subportadora que trabaja a una frecuencia de 4.43MHz (Europa) o a 3:58MHz (en USA, LA y Japón).

Cada línea explorada contiene un breve tren de impulsos, conocido como burst, muy cuidadosamente enfasado en la subportadora, colocado inmediatamente después del impulso de sincronismo de línea, que ha instruido al receptor a comenzar con una exploración. El álgebra puede entonces decodificarse de forma que pueda deducirse el verde: puesto que conocemos el valor de rojo, azul y de la luminancia total, el verde desconocido será: G = Y - (R+B).

Los tres sistemas de color que se utilizan actualmente, emplean distintos procedimientos de codificación y decodificación de los colores siendo completamente incompatibles.

El primer sistema inventado aún se utiliza en Estados Unidos, Latinoamérica, Japón y otros países. es conocido como NTSC de acuerdo con el National Television Systems Committee de 1953, el cual sufre todavía de una pobre tolerancia a la distorsión de fase de cada etapa del proceso, dando lugar al poco grato apodo Never Twice Same Color (nunca dos veces el mismo color).

En el sistema PAL (Phase Alternation Line) que es el estándar europeo (excepto para Francia), una de las dos señales de color cambia de fase cada línea alterna, mientras que la otra no. Por comparación de las fases en el receptor, cualquier distorsión y error de transmisión pueden eliminarse y mantenerse cuidadosamente el croma de los colores.

SECAM (Sequentiel Couleur a Mémoire), el sistema francés también adoptado por los rusos, tiene un principio muy diferente. La señal R-Y se transmite en una línea y la B-Y en la siguiente.

Un circuito de memoria en el receptor compara las dos y la estabilidad de color resultante permite el control total del color en el punto de transmisión.
Los estándares de líneas son NTSC 525, PAL 625 y SECAM 625; todos los sistemas pueden visualizarse en monocolor, en un receptor de blanco y negro.

2.2 SEÑAL DE VIDEO COMPUESTA.

2.2.1 GENERALIDADES.

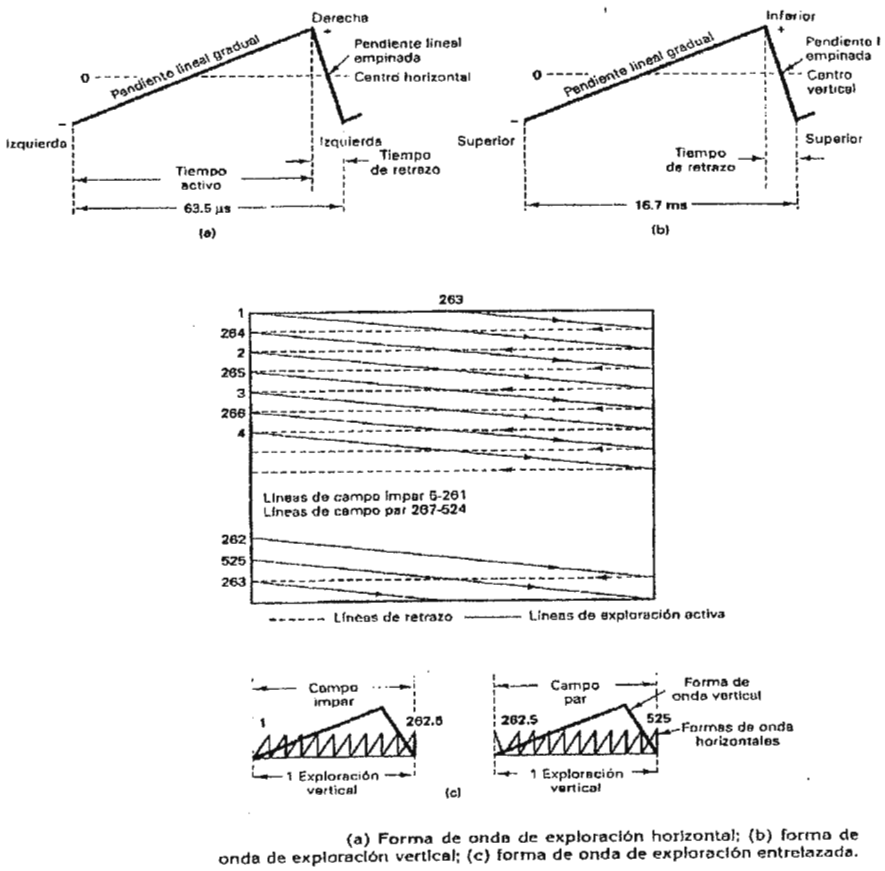
La señal de video está compuesta de valores analógico en la cual los valores más altos de voltaje producen trazos brillantes y los valores bajos representan el negro. La información de video se limita a frecuencias a 4MHz (20% menos que el sistema europeo PAL), por lo que su muestreo debe de ser alrededor de 4 veces mayor para que sea posible una reconstrucción completa. La señal de video contiene un cierto número de datos de información, para el receptor, además de la imagen en sí: cuando comenzar a explorar (impulso de sincronismo de línea), cuando retroceder a lo alto de la pantalla (impulso de sincronismo de campo) y debe dársele también un recordatorio, línea por línea, del nivel de negro puro que actúa como referencia para el grado de luminancia.

2.2.2 FORMAS DE ONDA DE EXPLORACION.

El haz de exploración para la cámara y el CRT en el receptor deben de moverse horizontalmente y verticalmente a razones uniformes. Esto se llama exploración lineal. La exploración lineal es necesaria para asegurar que los elementos de imagen no sean "aplastados" junto o "amontonados", a un lado o hacia arriba o hacia abajo de la imagen. La deflexión magnética suele utilizarse para mover el haz de electrones. Con la deflexión magnética, un alza lineal en la corriente por medio de bobinas de deflexión origina un cambio lineal en el flujo magnético. La fuerza del flujo magnético jala al haz de exploración de izquierda a derecha y de arriba a abajo de la pantalla en un movimiento continuo y uniforme. A continuación se muestra una onda de

exploración horizontal lineal. La inclinación positiva mueve el haz de izquierda a derecha en forma suave y constante (debido a su pendiente pequeña). La inclinación negativa de la forma de onda genera un campo magnético con la polaridad opuesta; por lo que el haz se mueve al lado izquierdo. Este es el tiempo de retraso y como se desea tan corto como sea posible su pendiente es más alta. Cuando fluye corriente cero, el haz está en el centro de la pantalla. cada línea de exploración tarda un ciclo de onda; por tanto, la frecuencia de esto es igual a la razón e exploración horizontal 15750 Hz y el tiempo para cada línea de exploración es de 63.5 microsegundos.

A continuación se muestra una forma de onda de lo antes descrito:



(a) Forma de onda de exploración horizontal; (b) forma de onda de exploración vertical; (c) forma de onda de exploración entrelazada.

Fig. 12 Señales de sincronismo horizontal y vertical.

Para reproducir la imagen original, las razones de exploración horizontal y vertical en el receptor deben de ser iguales a las de la cámara. Además, las líneas de exploración en la cámara y el

receptor deben iniciarse y terminar en una sincronización de tiempo exacta. En consecuencia, un pulso de sincronización horizontal de 15750Hz y un pulso de sincronización vertical de 60Hz se agregan a la señal de luminancia en el transmisor.

Los pulsos de blanqueo (blanking) son señales de video que se agregan a la luminancia y a los pulsos de sincronización con la amplitud correcta para asegurar que el receptor sea blanqueado durante los tiempos de retraso verticales y horizontales. La imagen no se explorará por la cámara durante el retraso y por consiguiente ninguna información de luminancia será transmitida para esos tiempos. Los pulsos de blanqueo en esencia "apagan" el CRT para los instantes en los que no hay imagen.

2.2.3 LA SEÑAL COMPUESTA.

La señal de video compuesta incluye señales de luminancias (brillantez), pulsos sincronizados horizontal y vertical y pulsos de blanqueo. La señal de pico a pico tiene entre 1 y 2Vpp, el valor exacto no es importante sino que el valor relativo es el que controla la brillantez. Estos valores de voltaje se han normalizado a 160 unidades IEEE. La máxima brillantez (blanco puro) es 120 unidades IEEE y para las señales menores al nivel negro de referencia no se genera brillantez alguna. El nivel negro de referencia se conoce también como pedestal o nivel de instalación negro. El nivel de blanqueo es 0 unidades IEEE, lo cual está abajo del nivel de negro.

En la siguiente figura se muestra la señal de video compuesta para el campo par, la cual tiene 16.7 ms y es el tiempo suficiente para 262.5 líneas de exploración horizontal. Sin embargo, el ancho del pulso de blanqueo está entre 0.05 y 0.08V con 833 a 1333 microsegundos. Por consiguiente, el pulso de blanqueo vertical ocupa el tiempo de 13 a 21 líneas de exploración horizontal, lo cual deja 241.5 a 249.5 líneas de exploración horizontal activas. La figura también muestra que casi todas las líneas de exploración se generan durante la inclinación positiva de la forma de onda de exploración vertical y el retraso vertical se presenta durante el pulso de blanqueo vertical.

A continuación se muestra un esquema ampliado de una de esas señales y una trama entera.

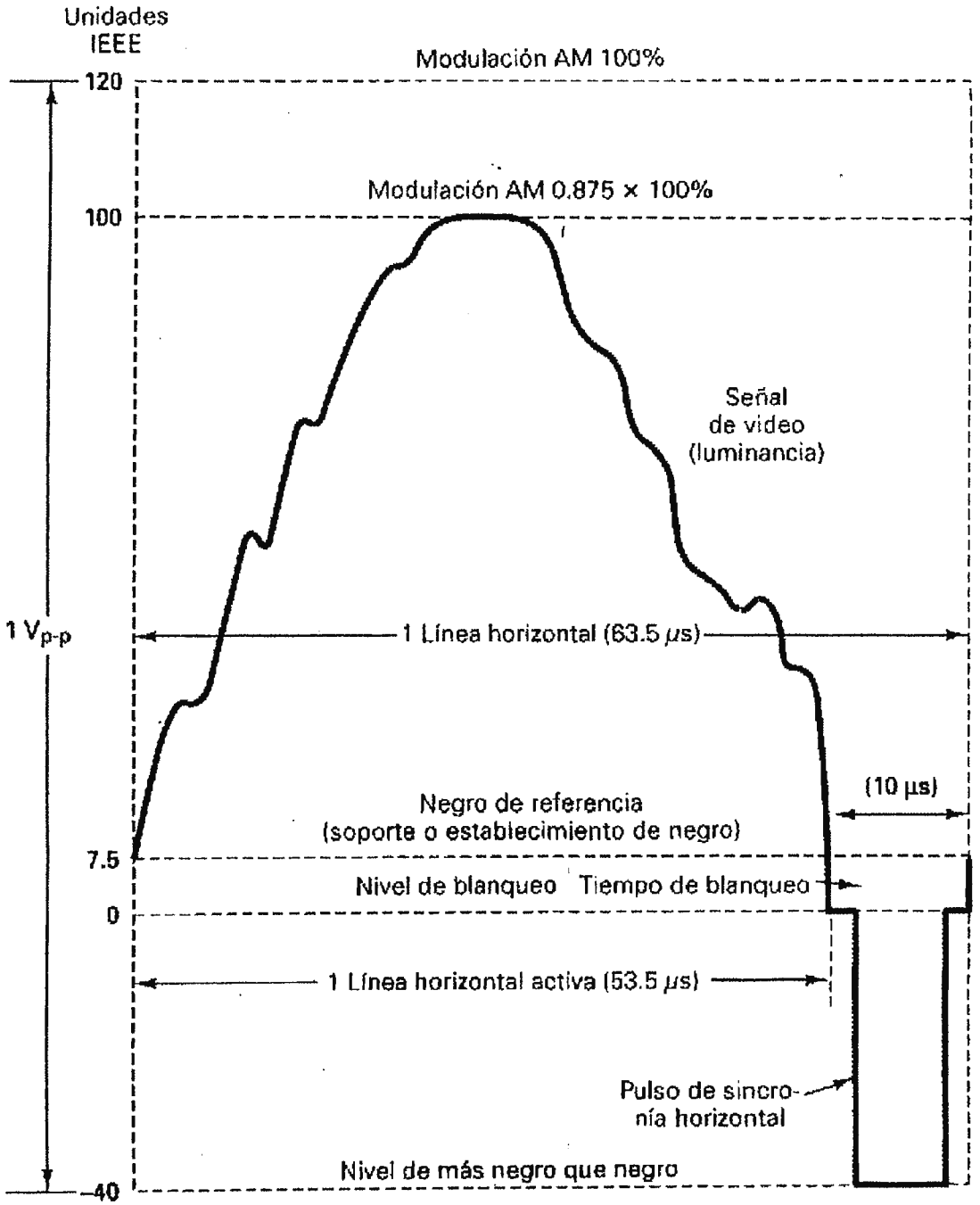


Fig. 13 Un pulso de información de señal de video compuesta.

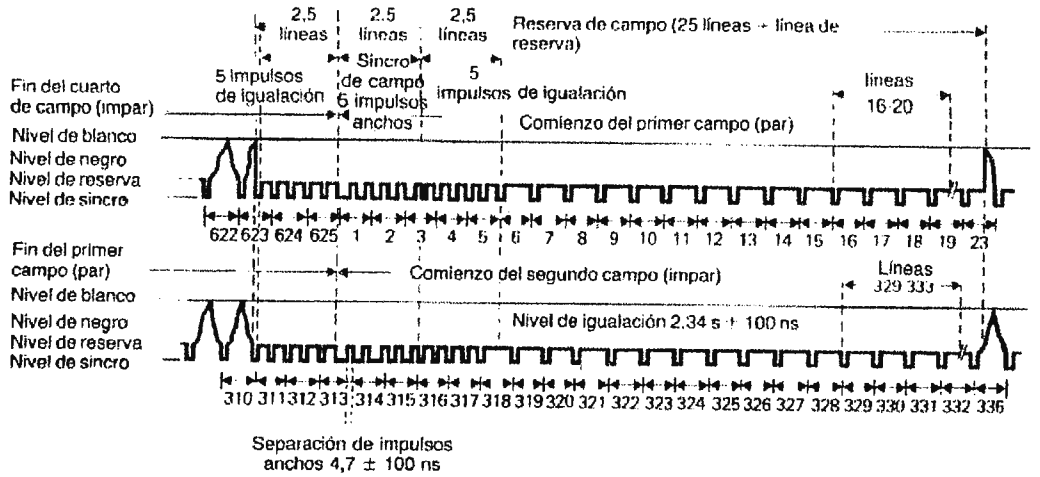


Fig 14. Trama completa de señal de video compuesta.

Los diagramas anteriores muestran precisamente los comienzos de dos campos sucesivos, línea por línea en un cuadro de 625 líneas. La imagen real de video no comienza hasta las líneas 23 y 326 respectivamente. Lo que se ve aquí es el impulso de sincronismo de campo y los impulsos para establecer el nivel de negro. La primera señal del campo de video correrá después desde las líneas 23 a las 310, en cuyo punto comienza el segundo campo (línea 311).

En la figura anterior se muestra el diagrama de una línea de video comenzando con el impulso sincro de línea, después el tren de datos y el nivel de negro, seguido por la señal de video para esta línea, la cuál en este caso toca el pico de blanco en dos puntos a través de la pantalla.

2.3 PROCESAMIENTO DE LA IMAGEN CAPTURADA.

2.3.1 TRATAMIENTO DEL RUIDO.

Las imágenes ruidosas pueden ocurrir debido a inestabilidad en la fuente de luz o en el detector de la imagen o algún error en la digitalización de la imagen o interferencias eléctricas del entorno. El ruido se puede describir como una variación parásita y aleatoria de la brillantez de la imagen, produciendo una alteración leve o severa.

Hay diversas maneras de tratar estas variaciones no deseadas y para el caso de esta tesis se usará el método de promediado para suprimirlo.

Esta técnica consiste en tomar una suma ponderada (es decir en la que se asignan un peso a cada valor) del pixel en cuestión con los de su vecindario. El resultado de esta suma ponderada es dividido entre la suma de los pesos dados a cada pixel y el resultado se coloca como nuevo valor del pixel. Esta operación es más eficiente si se tiene la imagen origen (a filtrar) en un buffer de memoria diferente al de la imagen resultado (ya filtrada) con el objetivo de que la operación se realice sobre los pixeles originales y no tome en cuenta a los ya modificados. Si los pesos asignados a cada pixel son 1 se está realmente obteniendo el promedio de los pixeles vecinos en el pixel a modificar. Es importante hacer notar que el pixel a modificar (el central del vecindario considerado) se incluye en la operación.

La ecuación general para estas sumas ponderadas (también llamadas máscaras) es:

$$P^{*}_{x,y} = \frac{\sum_{i,j=-m}^{+m} W_{i,j} \cdot P_{x+i,\,y+j}}{\sum_{ij=-m}^{+m} W_{i,j}}$$

Ec. 1. Regla de reasignación de valor de un pixel en base a su vecindario.

El vecindario puede ser de 3x3, 5x5, 9x9 o etc.; conforme aumente el valor de vecindario (más pixeles incluidos en la operación) se reduce el ruido presente en la imagen, pero se vuelve difusa

la imagen por el hecho de que esta se "normaliza" ya que los pixeles tienen valores más cercanos. Dicho de otra forma, la imagen pierde sus detalles como contornos y líneas delgadas o muy juntas.

El ocupar una máscara como:

1	1	1
1	1	1
1	1	1

Daría a la imagen un buen filtrado pero tendería a volverla muy difusa, debido a que el pixel central (a ser modificado) tiene el mismo peso que su entorno (3x3 en este caso).

Si se usará una máscara como:

1	2	1
2	4	2
1	2	1

La imagen no perderá tantos detalles pero la reducción del ruido no será tan eficiente.

A este filtro se le conoce como paso bajas.

Existen otros tipos de filtros que realizan operaciones con el entorno del pixel, tal es el caso del filtro de mediana. Este método consiste en colocar en el pixel a modificar el valor de brillantez que se encuentre a la mitad de todos los valores de brillantez listados del vecindario. Esto quiere decir, que en todo momento un pixel del entorno dará el valor del pixel a modificar. Este tipo de filtro es especialmente útil en aquellas imágenes que tengan pixeles corruptos con valores extremos o en donde se pierdan valores de pixeles en la adquisición. Estos valores adquirirán el valor de la mitad de todo su vecindario o entorno.

Este filtro tiene la deseable característica de preservar los contornos de las imágenes, es decir que no se vuelve difusa la imagen filtrada.

Algunos de los vecindarios que se utilizan con estos filtros son los siguientes:

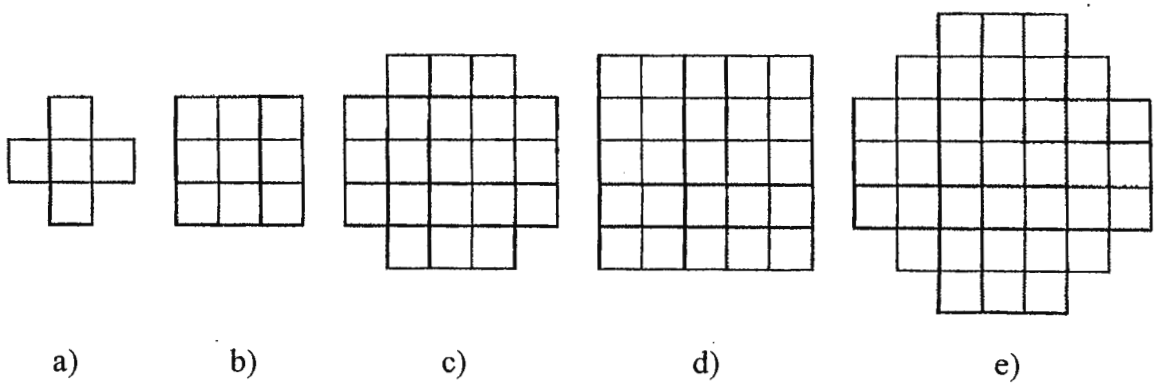


Fig.15 Diferentes vecindarios para las máscaras de los filtros: a) 4 en cruz, b) 3x3, c) 5x5 con 21 pixeles, d) 5x5 con 25 pixeles, e) 7x7 con 37 pixeles

2.3.2 USO DE LA FUNCION DE BACKGROUND CUANDO EXISTE UNA ILUMINACION NO UNIFORME.

Una de las formas más útiles y usadas para diferenciar objetos dentro de una imagen y así poder clasificarlos, es por medio de su nivel de gris. Para que esta técnica sea efectiva se debe de tener el mismo nivel de brillo en todo el cuerpo, lo cual no podrá conseguirse si la iluminación no es uniforme. Si esta iluminación llega a ser extremadamente no uniforme se conseguirá que la cámara capte al objeto con una forma totalmente diferente de la que realmente tiene debido a la diferencia exagerada en los pixeles y a sombras sobre el cuerpo mismo.

Esta iluminación no uniforme en la imagen puede ser modificada de manera física: colocando diversas fuentes de luz, o colocando una fuente de luz que proyecte sus haces de manera uniforme. También se debe alterar la superficie de los objetos para modificar su reflexión.

Una manera de corregir esta no uniformidad por medio de software es realizando una sustracción de una imagen a partir de otra para consiguen que las partes que tengan discrepancias muy grandes con el resto tiendan a uniformizarse. Esto trae de manera inevitable la pérdida de cierto datos de la imagen original, lo cual justifica que primero se debe de buscar la manera de uniformizar de manera física (antes de hacerlo por software).

Todos los pixeles de una imagen pueden ser listados dando como valores de clasificación su coordenada (x, y), y su valor de brillo. Teniendo esos valores listados, se pueden introducir a una función B(x,y) que pueda aproximar al fondo de la escena (o "background"). Luego se le resta a cada valor de la imagen adquirida su correspondiente valor en B(x,y) y se debería obtener como resultado el valor de fondo que "debería" ser.

La función de normalización o "background" (como también se le conoce debería tener la siguiente forma:

$$B(x, y) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 y^2 + a_5 xy$$

Ec. 2. Función de uniformización de fondo o de "Background".

Esta función polinomial tiene seis constantes fijas, que deberían de ser obtenidas a partir de 6 puntos del fondo de la escena proporcionados anteriormente y que constituyan una muestra razonable de los valores a los que se desea uniformizar. Este muestreo de puntos se puede hacer de manera repetitiva por software para elegir de entre diferentes grupos de 6 pixeles de muestra los que arrojen los resultados más aproximados. Otra manera de aplicar esta función de background es ir seleccionando por rectángulos (o pedazos) el fondo de la escena, ir seleccionando en ese rectángulo los 6 pixeles de muestra y a partir de ellos realizar la operación con la función solamente en esa área seleccionada.

2.3.3 DISTORSION POR LA PESPECTIVA.

En toda imagen se obtiene una representación x, y de un mundo que es X, Y, Z; dicho de otra forma el mundo real es tridimensional es 3D, y las imágenes son 2D. Esto trae consigo inevitables deformaciones debido a la profundidad de los cuerpos (algo que sucede aún con nuestra vista). A estas deformaciones se les denomina: *perspectiva.*

Matemáticamente el problema se puede solucionar de la siguiente manera: se obtienen un par de ecuaciones X', Y' en la que se presentan las coordenadas dentro de la imagen deformada por la perspectiva con respecto a las coordenadas X, Y que tiene el mundo real. Estas ecuaciones no deberían de contener solamente términos lineales sino también términos cuadráticos o XY, para

obtener todas las deformaciones posibles. Para simplificar las ecuaciones se excluyen los términos de valores mayores a 1 y se dejan los XY que proporcionará la dependencia entre ambas coordenadas (horizontal y vertical).

Las ecuaciones deberán ser como se muestran a continuación:

$$X = a_1 + a_2 X' + a_3 Y' + a_4 X' Y'$$

$$Y = b_1 + b_2 X' + b_3 Y' + b_4 X' Y'$$

Ec. 3 y 4. Coordenadas X, Y (respectivamente) del mundo 3D obtenidas a través de una imagen 2D que presenta distorsión por la perspectiva.

Los valores X, Y son las coordenadas del mundo 3D representadas en la imagen y las coordenadas X' , Y' (que son conocidas) se encuentran en la imagen. Para resolver esta ecuación se necesitan cuatro puntos de los cuales se tenga una correspondencia conocida con la realidad. Estas ecuaciones tienen sólo cuatro términos, aunque en la realidad los polinomios son de mayor orden. Este truncamiento hace un equilibrio entre exactitud y simplicidad.

En la práctica resulta más conveniente usar un método diferente a esas ecuaciones. El método consiste en saber que la deformación será siempre trapezoidal (si no hay rotación) y en que se pueden trazar líneas imaginarias dentro de toda la imagen. Un ejemplo de estas líneas imaginarias (que también pueden ser físicas) se muestra en la siguiente figura:

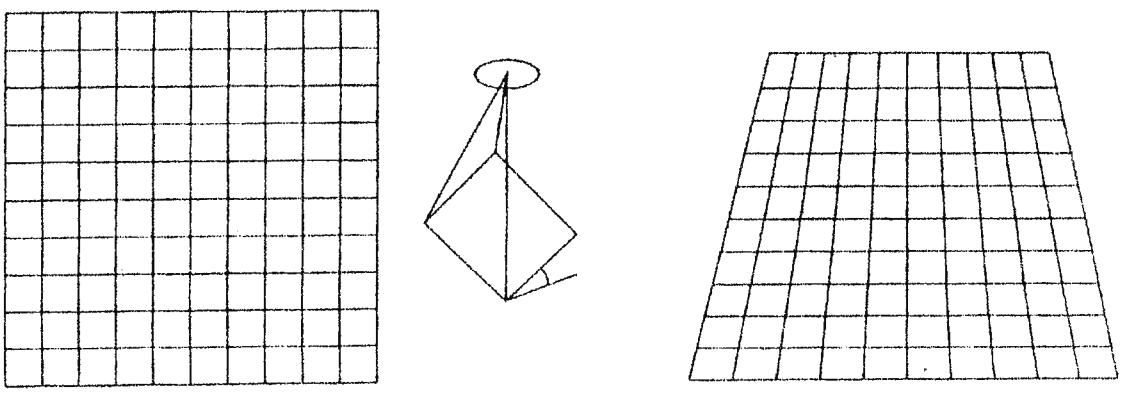


Fig. 16 Deformación trapezoidal debido a la perspectiva.

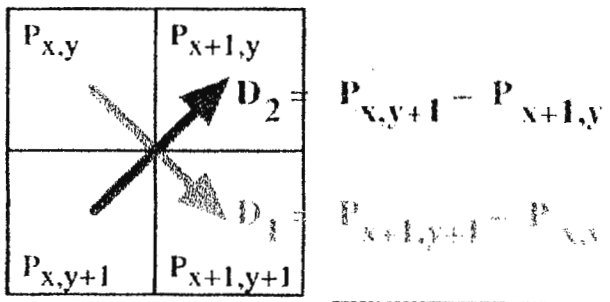
Como puede verse en la imagen distorsionada, las líneas horizontales de la cuadrícula corresponden también a líneas horizontales en la imagen, cuyos espacios entre ellas se reducen a la distancia. Esto hace necesario obtener una escala a partir del número de pixeles entre línea y la distancia que existe en el mundo 3D entre esas líneas. La situación se complica más en la vertical debido a que cada línea vertical en el mundo real tienen la misma distancia, sin embargo, en la imagen estas líneas adquieren una inclinación que aumenta conforme se alejan del centro. Esto se puede solucionar recordando que entre una línea vertical en la imagen y la línea de referencia inclinada existe un ángulo que es constante. A partir de esto se puede aplicar trigonometría para solucionar el problema.

2.3.4 OPERADOR SOBEL Y KIRSCH.

Los métodos de derivadas se basan en aplicar máscaras al vecindario del pixel y a partir de esto modificar su valor. Es muy importante que la imagen resultante y la imagen fuente se encuentren en diferentes buffer de memoria. Los operadores de Sobel y Kirsch usan estos mismos principios de las derivadas y siendo uno de sus mayores aplicaciones el permitir esqueletizar.

Se basa en usar los valores máximo y mínimo en la derivada de la imagen para colocar los valores en blanco (los bordes) o en negro (el resto de la imagen). Se puede también definir una escala de tonalidades de gris, cuyo valor de brillantez dependa del grado de variación de la brillantez del vecindario considerado.

Si las derivadas en las direcciones ortogonales, que no necesariamente son vertical y horizontal, son calculadas se pueden combinar sacando su magnitud como se muestra a continuación:



$$\text{Result} = \sqrt{D_1^2 + D_2^2}$$

Fig. 17 Diagrama de la Cruz de Roberts en la que se basa el operador de Sobel.

$$\text{Magnitude} = \sqrt{\left(\frac{\partial B}{\partial x}\right)^2 + \left(\frac{\partial B}{\partial y}\right)^2}$$

Ec. 5 Magnitud en la que se basa el operador de Sobel.

Este es el método de Sobel (1970) y las derivadas mostradas en la figura se conocen como la Cruz de Roberts.

Uno de los métodos más usados cuando las operaciones matemáticas para calcular la raíz cuadrada realizan retardo (por no existir un procesador matemático) es el método de Kirsch (1971). Este método aplica cada una de las 8 orientaciones posibles de las derivadas y se guarda el valor mayor. Esto requiere utilizar máscaras como las que se muestran a continuación:

1	0	-1
2	0	-2
1	0	-1

2	1	0
1	0	-1
0	-1	-2

1	2	1
0	0	0
-1	-2	-1

Este operador da casi los mismos resultados que los de Sobel y su facilidad de realización por software es mucho mejor.

2.3.5 SUSTRACCION DE IMAGENES.

Como se mencionó en la técnica de la función de "background" se puede restar una imagen de otra para determinar las diferencias entre dos imágenes. En una operación de sustracción, el rango posible de valores (en una escala de grises) es de 0 a 255, llegando a ser el nuevo rango de -255 a +255. El dato resultante puede ser re-escalado a un sus valores correctos, sumando 255 y luego dividiendo entre 2 (luego se redondea el resultado). Una de las aplicaciones principales de la sustracción de imágenes es para resaltar las diferencias entre la misma imagen después de que se le ha procesado. La idea es obtener en la imagen resultante solamente aquellas partes (pixeles) que cambian y eliminar (o uniformizar) todo aquello que no cambie.

La sustracción de imágenes se puede utilizar para control de calidad, de tal manera que la imagen tomada del producto se compare con una imagen de referencia. Mientras existen más diferencias la calidad de la pieza producida se asume menor. En la siguiente figura se muestra el uso de la sustracción para determinar si un objeto está alineado o no:

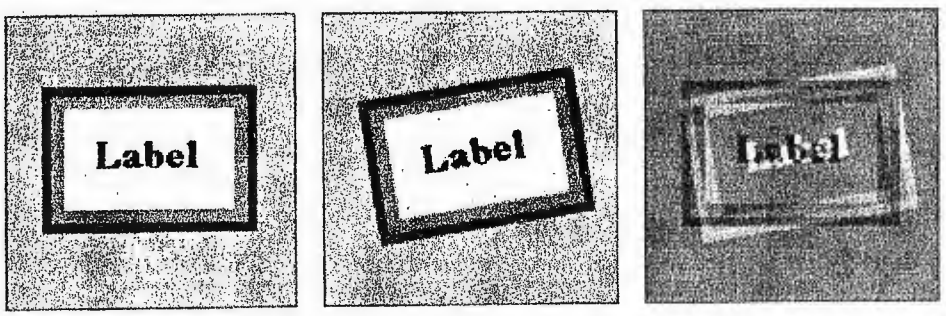


Fig. 18 Uso de la substracción de imágenes para detectar malos alineamientos.

2.3.6 UMBRALIZADO.

Seleccionar las figuras dentro de una escena es uno de los más importantes prerequisitos para la mayoría de las mediciones que se realicen dentro de la escena. Tradicionalmente, un camino simple para el umbralizado ha sido definir un rango de valores de brillo en la imagen original, colocarlo a un valor predeterminado y todos los demás referirlos al valor de brillo del fondo. De esta forma la imagen ahora contendrá solamente dos valores dentro de la imagen (que cuando son blanco y negro se le denomina binarizado).

El umbralizado pueden ser usados para generar una nueva imagen en la cual el brillo representa algún parámetro de derivadas como el gradiente o la dirección.

Otra forma de realizar el umbralizado es colocar un rango de grises que cambiarán a un valor predeterminado y si no se encuentra el pixel en ese rango se coloca a otro valor muy diferente del que cumple con la condición. Esta es la técnica usada en la presente tesis.

Es interesante señalar que se puede realizar una umbralización de multinivel. En este caso, se definen diversos rangos no traslapados de grises y cuando un pixel cumplan con ese rango cambiará a un valor de pixel predefinido. Estos rangos son, generalmente, angostos y ninguno de lo nuevos valores de pixeles es compartido por ninguno de los rangos. Este umbralizado de múltiple nivel se utiliza para diferenciar diversas figuras a partir de su tonalidad de gris. Esta técnica no dará buenos resultados si la imagen tiene iluminación no uniforme, ya que las diferencias de luminosidad del entorno se traducirán en valores totalmente distintos de las escalas de grises, lo que se conseguiría no es diferenciar figuras; sino aumentar la no uniformidad.

Por medio de umbralizado, se consigue distinguir las figuras del fondo de la escena. El aplicar métodos analíticos a imágenes umbralizadas es más sencillo que el hacerlo en aquellas con muchos niveles.

Una desventaja de la aplicación del umbralizado es que objetos que tengan el mismo nivel de gris y se encuentren traslapados se tendrá en la imagen una representación de ambos como si fueran uno.

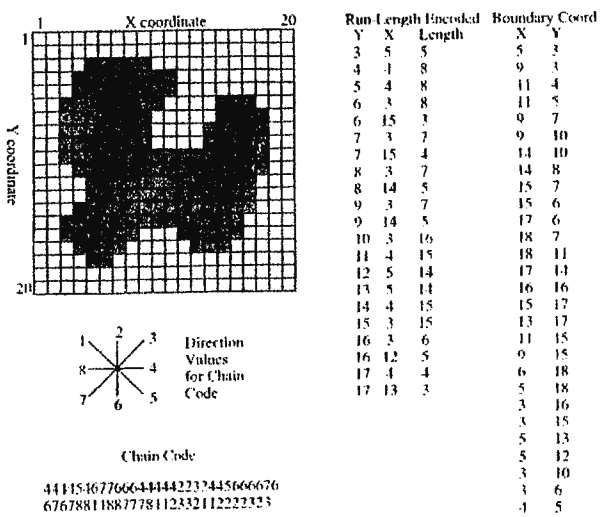


Fig. 19 Representación de una figura dentro de una imagen, con pixeles aumentados.

2.3.7 LINEAS DE CONTORNO.

Un tipo de línea que puede proveer información del entorno y de la que se está seguro de su continuidad, es la línea de contorno. Esto es análogo a las líneas de iso-elevación dibujadas en los mapas topográficos. La línea marca una elevación constante en esos mapas, en nuestro caso las líneas representan un mismo nivel de gris.

La línea de contorno puede en principio se aproximada por un polígono, para hacer que el contorno se vuelva suave se puede interpolar puntos entre la línea del polígono y trazar otras líneas, lo cual aumenta el número de lados del polígono. Al tener más lados, en el infinito (conceptualmente), la línea se volverá una curva de trazo suave.

Cuando las imágenes se hacen a dos niveles de gris (que puede ser negro y blanco), se puede seguir un algoritmo diferente a los operadores de Sobel y Kirsch antes citados. El método consistiría en recorrer todos los pixeles de la imagen de tal manera que se le represente por líneas horizontales de pixeles. Los valores extremos de cada una de estas líneas son el contorno lateral y puede ser detectados por el hecho de que entre ellos se produce una discontinuidad en el nivel de gris. Se guardan estos valores en memoria y luego se cambian en la imagen resultante todos los demás pixeles que no estén a los lados. La siguiente figura puede ayudar a vislumbrar la manera de como funciona este algoritmo. El proceso se puede repetir para las columnas y al final se tendrá todo el contorno.

2.3.8 OPERACIONES BOOLEANAS.

Las imágenes binarias pueden ser "enmascaradas" por otras imágenes binarias. Este "enmascaramiento" se puede hacer con las funciones booleanas AND, OR y NOT convencionales. Estas operaciones se pueden usar para ocultar figuras, uniformizar regiones de la imagen y aún resaltar diferencias mínimas (como pequeños desplazamientos) entre dos tomas diferentes de la misma imagen.

A continuación se muestra diversos ejemplos de operaciones booleanas aplicadas a imágenes:

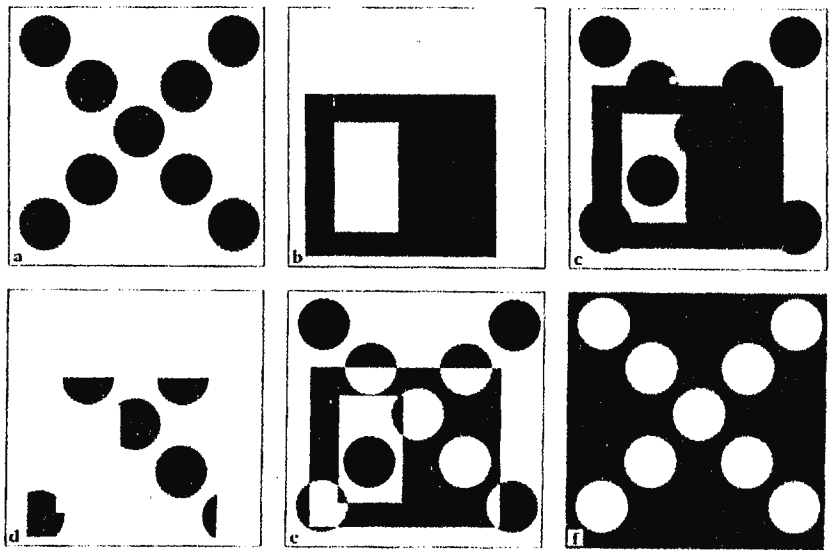


Fig. 20. Operaciones booleanas simples: a) y b) imagen binarizada, c) A or B, d) A and B, e) A exor B, f) not A.

El valor de ON se puede asignar ya sea al valor más oscuro o más claro de los dos valores posibles dentro de la imagen binarizada. El programador debe de elegir cual será el valor correspondiente y el sobrante es OFF. Las operaciones booleanas se hacen por lo general entre dos imágenes binarizadas.

Una AND requiere que ambos pixeles correspondientes en las imágenes tengan el valor de ON para que el resultado se coloque en ON, en todo otro caso será OFF. La operación OR colocará un valor de ON en la imagen resultante si al menos uno de los pixeles correspondientes en las imágenes tiene la condición de ON. La operación EXOR colocará la salida en ON si de los pixeles comparados en cada imagen, solamente una de las imágenes la tenga en ON; si ambos pixeles son iguales se tiene OFF. La más básica de todas es la NOT, que simplemente resta a 255 el valor actual y esa diferencia se asigna como nuevo valor al pixel. Esta operación sólo necesita de una imagen.

En ciertas aplicaciones estas operaciones no son suficientes y lo que se hace es combinar diferentes operaciones para obtener el resultado deseado. A continuación se muestran algunas imágenes de ejemplo:

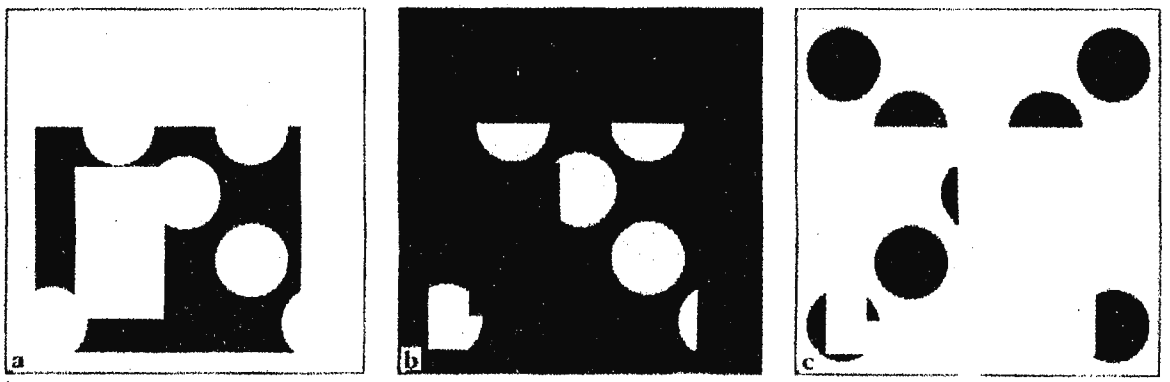


Fig. 21 Operaciones booleanas combinadas : a) (not A) and B, b) not (A and B), c) A and (not B).

2.3.9 EROSION Y DILATACION.

Esta operación se clasifica dentro de los que se conoce como operaciones morfológicas. Se aplican a imágenes binarizadas. Estas operaciones a nivel de software se basan en detecciones de vecindarios (generalmente 3x3) en lo que se realizan cambios entre el valor asignado al fondo de la escena y el valor dado a los objetos. Estas operaciones se pueden aplicar de manera repetitiva. Ambas operaciones pueden ser descritas en términos de adición o remoción de pixeles desde la imagen binaria de acuerdo a ciertas reglas, las cuales dependen del patrón en el vecindario de pixeles.

La Erosión remueve los pixeles desde una imagen, colocando en OFF los valores que se encuentran en ON en la proximidad inmediata de pixeles en OFF. El efecto primero que se notará es que los objetos pequeños y aislados se perderán y aquellas regiones angostas de los cuerpos se cortarán. Todos los pixeles de la periferia de un objeto serán colocados al mismo valor que tenga el fondo.

La dilatación se basa en un principio inverso. Cuando se encuentran los valores que corresponden a las transiciones entre figura y fondo de escena se colocan en ON todos los pixeles OFF que se encuentren junto a los valores del contorno del cuerpo. El efecto resultante es que los pequeños puntos en la imagen se agrandan, los agujeros dentro del cuerpo se pierden y todas las figuras se hacen más "gruesas".

En la discusión anterior la expresión ON se refiere al nivel de gris asignado a los objetos dentro de la escena.

A continuación se muestra la aplicación de las operaciones antes descritas:

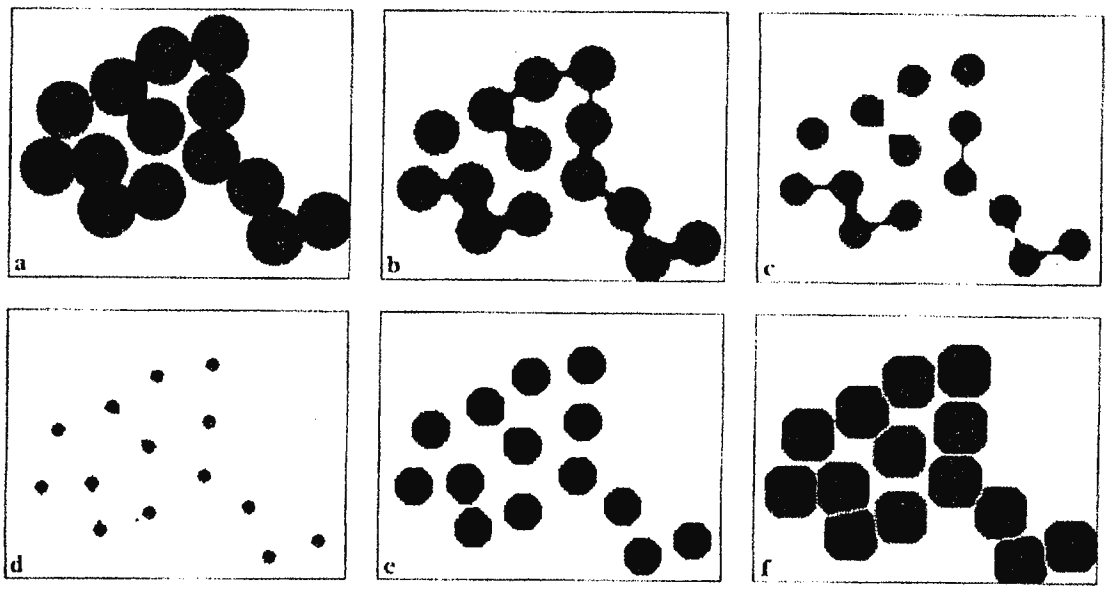


Fig 22. Separación de figuras por medio de erosiones y dilataciones sucesivas. Desde b) hasta d) son erosiones y desde e) hasta h) son dilataciones.

2.3.10 APERTURA (OPENING) Y CERRADURA (CLOSING).

La combinación de una erosión seguida por una dilatación es llamada una apertura, referida por la habilidad de ampliar los espacios entre las imágenes que se están cercanas. Esta es una de las secuencias más comúnmente utilizadas para remover ruido de las imágenes binarias.

La operación de cerradura realiza las mismas operaciones pero en orden inverso. Una de las aplicaciones clásicas de esta operación es la de rellenar rupturas en la imagen o para juntar porciones de un mismo cuerpo que se han separado. Un ejemplo de aplicación de un operación de cerradura se muestra a continuación:

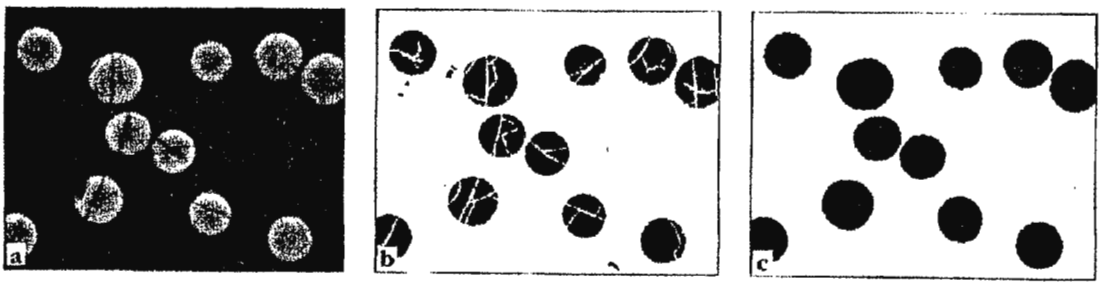


Fig. 23 Reconstrucción de figuras por medio de closing: a) imagen original, b) imagen binarizada, c) imagen reconstruida después del closing.

Una aplicación interesante de estas operaciones puede ser la siguiente:

Si se tiene una imagen binarizada, se le puede aplicar una erosión y se guarda el resultado en un buffer de memoria o en disco. Luego a la misma imagen original binarizada (sin aplicar la erosión) se le aplica una dilatación y se guarda en un buffer de memoria diferente que el anterior. Después se toman ambas imágenes y se les aplica una EXOR entre sí. El efecto total es que en la resta quedarán solamente los bordes exactos de las figuras. Esto de debe a que la dilatación aumento las dimensiones de la figura y la erosión las contrajo; la EXOR dejará prácticamente la diferencia, que en este caso serán los bordes.

Lo anterior se muestra en las siguientes figuras de ejemplo:

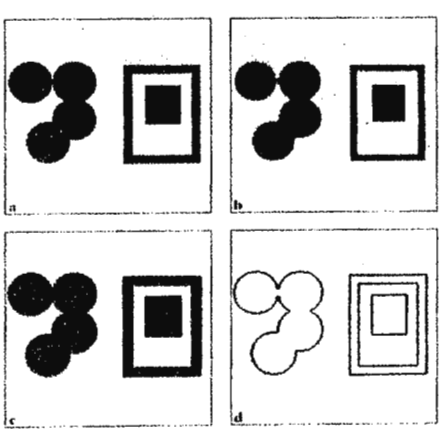


Fig. 24 Ejemplo de aplicación del closing, opening y las operaciones booleanas: a) imagen original, b) imagen erosionada, c) imagen dilatada, d) exor de las dos anteriores.

2.3.11 ESQUELETIZACION

La esqueletización es el acto de realizar las operaciones necesarias para dejar solamente las líneas que forman la imagen, incluidas las del contorno y la de los detalles internos de las figuras.

Esta operación se puede aproximar por el uso de los operadores Sobel y Kirsch antes expuestos. También puede ser realizada por medio de una erosión efectuada con reglas especiales que la remoción de pixeles, deteniéndose el proceso cuando se tienda a partir la región en dos. La regla para hacerla es examinar los vecindarios tocantes: si ellos no forman un grupo continuo, entonces el pixel central no puede ser removido (Pavlidis, 1980). La definición de esta condición es dependiente de si la conectividad es de 4 u 8 pixeles, esto se muestra en las siguientes figuras:

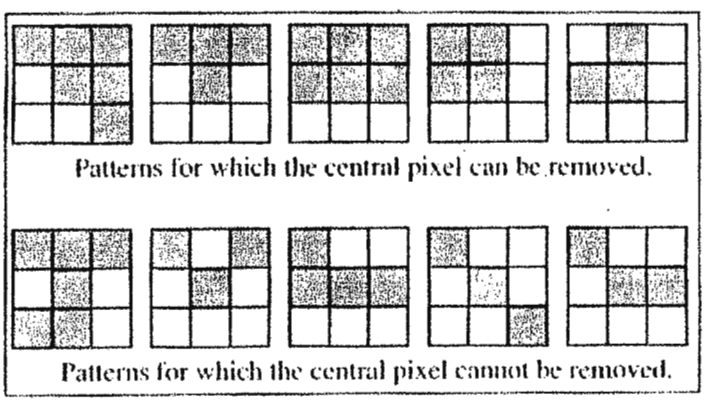


Fig. 25 Representación de los patrones de vecindario que permiten y no permiten al pixel central ser removido en la esqueletización.

La esqueletización permite clasificar las diferentes figuras presentes en la escena, facilitando también algunas mediciones como longitudes, ya que todo se representa por líneas. Otra ventaja de la esqueletización es la adelgazar figuras delgadas, en las cuales sus líneas de contorno aparecen gruesas.

El problema con la esqueletización es su sensibilidad para detectar como cambios en la forma pequeñas discontinuidades en la figura. En ocasiones puede suceder que la línea aparezca en la

esqueletización como si estuviera cortada o también podría aparecer la línea en una dirección que no es la correcta. También se presenta el defecto de que si el cuerpo en la escena es 3D y tiene una textura con líneas transversales se pueden generar líneas fantasmas que podrían indicar que la figura está formada por muchas otras, aunque sea una sola.

2.3.12 MAPA DE LA DISTANCIA EUCLIDEA.

El mapa de la distancia euclidea (Euclidean Distance Map o EDM) es una herramienta que trabaja en una imagen binaria para producir una escala de grises. La definición de esta operación es simple: cada pixel, ya sea de la figura o del fondo de la escena, es asignado un brillo igual (o proporcional) a su distancia desde el límite más cercano. Esta es una manera de codificar (en el nivel de gris de cada pixel) la distancia de cada pixel a su límite más próximo.

Estas unidades de distancia se pueden realizar de diferentes formas, ya que se pueden tomar por distancia de pixeles, dando como válidas distancias a 45 grados o calculando directamente por el teorema de Pitágoras. Lo anterior se ejemplifica en una simulación de bits un mismo vecindario con sus diferentes valores de distancia y sus nuevos valores de gris.

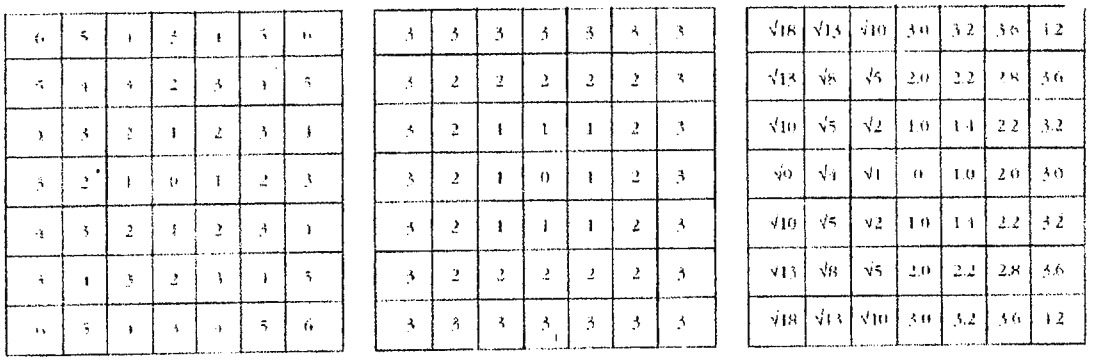


Fig. 26 Arreglo de pixeles con su distancia desde el pixel central mostrando diferentes casos de asignación de las distancias.

Un algoritmo expuesto de forma conceptual para realizar esta operación, podría ser el siguiente:
Asigne un brillo de cero a cada pixel que forme parte del fondo.

- Coloque una variable N a un valor igual a cero.
- Por cada pixel que toque (en el vecindario de 4 u ocho) de quien su nivel de gris sea N, asígnele el valor de N+1.
- Incremente N y repita la asignación anterior hasta que todos los pixeles de la imagen hayan sido modificados.

El tiempo requerido para esta iteración puede ser relativamente grande. En 1980, Danielsson desarrolló un método más eficiente que el anterior para realizar lo mismo:

- Asigne un nivel de gris de cero a cada pixel en el fondo y un valor muy grande (que puede ser el máximo) a cada pixel en la figura de interés. Esto equivale a binarizar.
- Proceda de izquierda a derecha y luego de arriba a abajo, asigne a cada pixel alejado del centro geométrico de la figura un valor más grande que el más pequeño valor de alguno de sus vecinos.
- Repita el paso anterior, procediendo de izquierda a derecha y luego de arriba hacia abajo.

Es importante notar que al reasignar los valores de pixeles a partir del punto que se desea focalizar, se pierde mucha información del objeto (u objetos) en si, pero el sacrificio es justificable si lo que interesa es tener en la imagen misma la información de la distancia hacia un punto específico de ésta.
Si la figura tiene una forma radial (no necesariamente un círculo) el punto de interés es el centro o un punto que se considere como tal. Esta técnica puede no resultar muy conveniente si la figura tuviera una distribución no radial, como sería el caso de una figura muy extensa o alargada.
El mapa de distancia euclidea se puede interpretar como una altura (semejante a los gráficos de topología), tal como se muestra en el siguiente gráfico:

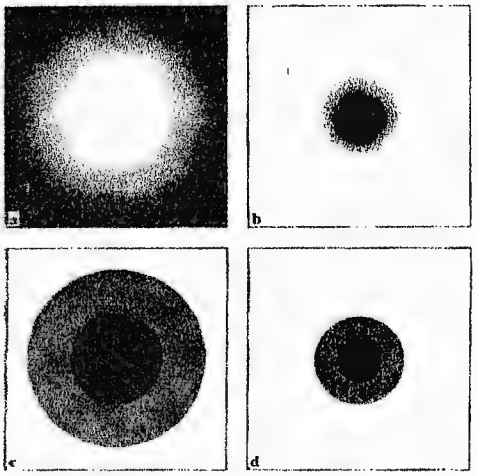


Fig. 27 Ejemplo de Mapa de la Distancia Euclídea (EDM). a) La EDM de el fondo alrededor del círculo, b) La EDM del circulo, c) Dilatación efectuada con un umbralizado del fondo en 50, d) erosión efectuada con un umbralizado de fondo de 25.

El propósito de la EDM es proporcionar una forma de dilatación y erosión a partir de un punto considera de interés, siendo particularmente útil en las figuras circulares. La dilatación y erosión antes descritas tienden a convertir en polígonos las formas circulares. A continuación se muestra una imagen de ejemplo:

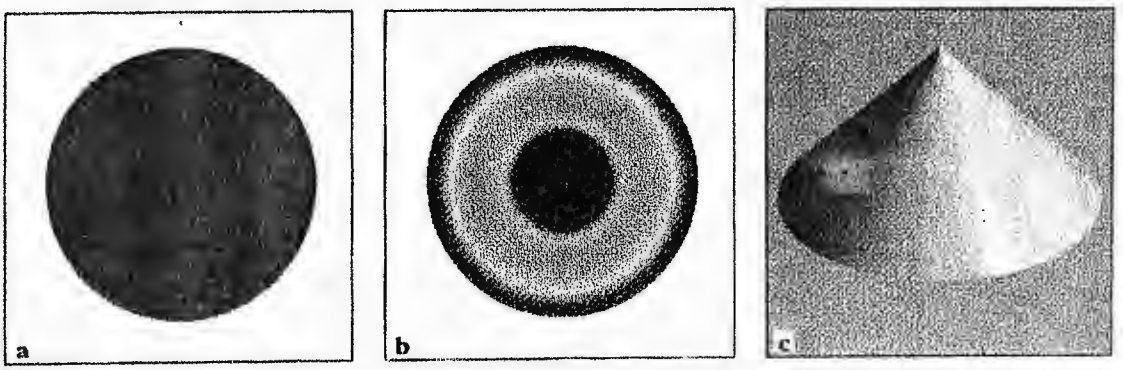


Fig. 28 Interpretación del mapa de la distancia euclidea: a)binarizada, b)con mapa de la distancia euclídea, c) forma obtenida al sacar datos a esa combinación, en base a condiciones de salida.

2.3.13 DETERMINANDO EL CENTRO GEOMETRICO.

Una de las necesidades mayores al realizar una segmentación es el tener la habilidad de para cuantificar la posición de una figura dentro de la imagen. Hay diferentes maneras de realizar la cuantificación de la posición de la figura: Una forma es tomando el punto medio a partir de los límites máximo y mínimo de coordenadas x, y que limitan al objeto (formando una caja alrededor del objeto), teniendo este método la desventaja de que no toma en cuenta de ninguna forma la distribución de los pixeles entre si.

Es importante señalar que los pixeles dentro de la imagen, tienen por naturaleza como coordenada de origen la esquina superior izquierda de la imagen. Sin embargo, con simples adiciones o substracciones se puede cambiar el origen del sistema de coordenadas x, y.

Anteriormente, se citó el caso de tomar el punto medio de las coordenadas que enmarcan (en un rectángulo) a la figura de interés, pero esta es una aproximación extrema. Un valor más razonable que puede representar en un punto a todo un objeto es la deteriminación de su centro de gravedad (C.G.), que para el caso en que exista homogeneidad coincide con el Centro Geométrico. Este centro de gravedad se puede calcular a partir de la siguiente expresión:

$$C.G._x = \frac{\sum_i x_i}{Area} \qquad C.G._y = \frac{\sum_i y_i}{Area}$$

Ec. 6 Ecuaciones para determinar el centro de gravedad de una figura que tiene un valor de pixeles uniformes.

Donde ***Area*** es el número de pixeles presentes en la figura. El valor del C.G depende del marco de referencia que se elija (sí es la esquina superior izquierda, el punto central de la imagen, la esquina inferior derecha, etc.); sin embrago su posición dentro del objeto es siempre la misma. De lo anterior se desprende que no importa el sistema de referencia en si, con tal que esa referencia se respete durante todo el proceso de cálculo.

Un detalle importante, es que los valores obtenidos de las operaciones anteriores son, en general, valores que contiene decimales y que se necesita hacer un cierto redondeo (según la necesidad del programador) pues los pixeles son enteros por naturaleza.

Traduciendo las expresiones generales anteriores a una expresión que pueda utilizarse para calcular regiones que no tienen valores homogéneos de pixeles, se puede convertir a:

$$C.G._x = \frac{\sum_i Value_i \cdot x_i}{\sum_i Value_i} \qquad C.G._y = \frac{\sum_i Value_i \cdot y_i}{\sum_i Value_i}$$

Ec. 7 Ecuaciones para determinar el centro de gravedad de una figura que está formada por pixeles de tonalidad de gris o color diferente.

Como puede verse los valores de brillantez de cada pixel influenciarán la ubicación del C.G.

Es interesante recalcar que los valores de xi y yi de estas ecuaciones son las simples posiciones x, y de cada pixel de la figura.

Es así, como ese punto del C.G. sirve para representar a la figura entera y facilitar su ubicación espacial en la imagen.

2.3.14 ORIENTACION.

Un concepto muy relacionado a la ubicación del C.G. es su orientación. Hay diferentes maneras de hacerlo: la primera forma es tomando la línea entre los puntos más distantes de la figura (como la punta y el borrador de un lápiz) y se calcula por simple trigonometría su ángulo o también calculando por estadística la dirección en que están distribuidos los pixeles. Sin embargo, estos métodos son difíciles de calcular o muy susceptibles a las distorsiones de la imagen.

La manera más adecuada de calcular la orientación (que por suerte tiene una expresión matemática) es por medio del eje del momento de inercia. Este eje tiene la orientación exacta

dada por la recta que está en la máxima longitud de la figura. Esta recta tiene un ángulo que puede ser calculado por medio de:

$$M_x = S_{xx} - \frac{S_x^2}{Area}$$

$$M_y = S_{yy} - \frac{S_y^2}{Area}$$

$$M_{xy} = S_{xy} - \frac{S_x \cdot S_y}{Area}$$

$$\Theta = \tan^{-1}\left\{\frac{M_{xx} - M_{yy} + \sqrt{(M_{xx} - M_{yy})^2 + 4 \cdot M_{xy}^2}}{2 \cdot M_{xy}}\right\}$$

Ec. 8 Ecuaciones para determinar el ángulo de orientación de una figura.

En donde se necesitan los siguientes valores previamente calculados:

$$S_x = \sum x_i$$

$$S_y = \sum y_i$$

$$S_{xx} = \sum x_i^2$$

$$S_{yy} = \sum y_i^2$$

$$S_{xy} = \sum x_i \cdot y_i$$

Ec. 9 Ecuaciones de las diferentes sumatorias de posiciones (x, y) de pixeles usadas comúnmente.

2.3.15 CONTEO DE FIGURAS.

El conteo del número de figuras presentes en una imagen o campo es una de los procedimientos mas comunes en el análisis de imágenes. La manera más sencilla de realizarla, es que por medio del punto de C.G. se representa de manera única a cada objeto detectado. Luego el conteo se

reduce a contar dentro de la imagen cuantos de estos puntos de C.G. hay. Cuando las figuras se interceptan (y aún se traslapan entre si) no es conveniente usar el procedimiento anterior, sino más bien se deben de contar las figuras que tocan dos orillas, por ejemplo la izquierda y la de arriba, ignorando las que tocan las otras orillas de abajo y la derecha. Este método es equivalente a contar cada figura por su esquina superior izquierda.

Este método es el mismo que determinar el número de personas en un cuarto por medio del conteo de narices. Cada persona tiene una nariz y esta tiene una forma muy bien definida. Aunque una persona pierda muchos detalles debido a que se traslape con otras, se supone que la búsqueda de las figuras humanas no será tan complicada debido a la forma constante de su nariz.

Hay otro método alternativo que consiste en usar la imagen entera pero no contar las figuras que tocan las orillas. Para evitar errores en el conteo se toma la proporción de la figura a contar, con respecto a la imagen entera, si esta sobrepasa las orillas no es tomada en cuenta; debido a los errores que puede traer contabilizar un objeto del que se tiene una vista parcial. Se puede hacer una fórmula de ajuste de conteo como se muestra:

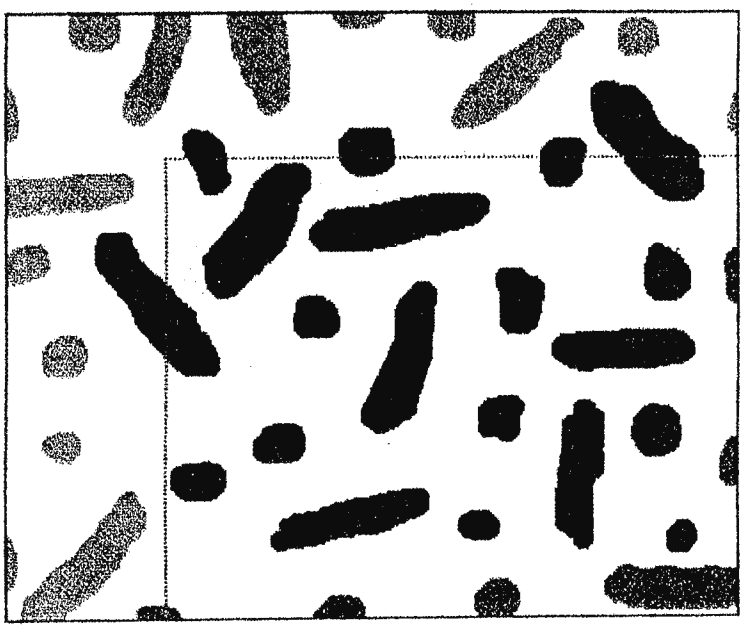


Fig. 29 Conteo de figuras por medio de regiones separadas.

Donde Wx y Wy son las dimensiones de la imagen en las direcciones "x" y "y" (pixeles), y Fx junto a Fy son las dimensiones de las proyecciones de las figuras en esas direcciones. Cuando las dimensiones de la figura son pequeñas comparadas con las dimensiones de la imagen (no total necesariamente sino del área de interés), la fracción anterior es cercana a 1.0 y el simple conteo no es afectado. Si la figura abarca mucho espacio de la región de interés la fracción se desviará mucho más de 1 y eso puede servir para no contarla o para realizar un cálculo que permita descontarla del conteo total (asumiendo que los conteos se van realizando por regiones). De esta forma las figuras que salgan de la imagen (o cubran varias porciones de imagen) no son contadas más de una vez.

2.3.16 MEDIDA DEL AREA DE UNA IMAGEN.

La manera más básica de diferenciar figuras entre es por medio de su área. Para un sistema basado en representaciones de pixeles, este es el número de pixeles dentro del contorno de la figura, los cuales se pueden tomar por un programa de conteo que detecte bordes.

Es bueno recordar que la medida de una figura en dos dimensiones puede ser relacionado a la medida de la correspondiente imagen 3D, que si se trata de la sección de una muestra vista en microscopio .puede tener realmente dos dimensiones. El más común tipo de imágenes son proyecciones, en las cuales las figuras son básicamente sombras de los objetos, o secciones planas de los mismos. En todo caso, lo más general es que se vea al objeto desde una dirección aleatoria, y es posible estimar el volumen del objeto desde esas proyecciones. Las reglas de estas estimaciones, las cuales se basan en probabilidad geométrica, son proporcionadas por la estereología. Las estimaciones son estadísticas en naturaleza, significando que el volumen de un objeto individual no es determinado exactamente, pero los objetos pueden ser clasificados en base a ello.

En otros casos lo que se desea es obtener una manera de diferenciar los objetos por su tamaño, por lo que no es muy crítico determinar su volumen. Es entonces cuando se recurre al área, que cuando los objetos tiene casi la misma profundidad se puede asumir como su volumen.

La manera de determinar el área depende mucho del propósito. Considere las siguientes figuras:

Fig. 30 Diferentes formas de medir el área de una figura irregular.

Surge una pregunta: ¿Se deben de tomar los agujeros para calcular el área?. La respuesta es depende. En al figura de la izquierda no se incluyen y debido las irregularidades del contorno se traza un circulo que cubra el promedio de la figura (el centro del circulo ese en el C.G.). Se cuentan los pixeles que se encuentran dentro del área del circulo y se suman representando eso el área. En la figura del centro se tiene el mismo circulo pero ahora se asume que los agujeros son parte de la figura, note que el área aumento al incluir en la suma los pixeles de los agujeros faltantes. La figura de la derecha tiene otra lógica y en este caso lo que se busca es que no sea un círculo la figura que circunscribe sino que un polígono. previamente se ha determina por programa las esquinas de cuerpo (que pueden ser redondeadas o cuadradas) y se empieza a sumar todos los pixeles de la figura que se encuentra dentro de ellos. Este método es mejor pero tiene la complicación de determinar sin errores las esquinas del cuerpo, que es general son cambios bruscos de dirección en el contorno (no asi para las figuras circulares). Este último es el método más exacto.

En todo caso la determinación del área de una figura es una aproximación de la realidad que se basa en un conteo de pixeles (previa detección de los contornos). Matemáticamente se expresa:

$$A=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} g(i,j)$$

Ec. 10 Cálculo del área de una figura.

2.3.17 DETECCION DE BORDES.

Se conoce como bordes a los límites de un objeto. Detectarlos es de suma importancia para realizar tareas de segmentación, registro e identificación de objetos en una imagen.

Las técnicas de detección de bordes se basan en la determinación de los puntos en que se produce una variación de la intensidad de gris del pixel (picture element). Los métodos para la detección de borde se basan en el operador derivada.

Con la detección de bordes se pretende realzar los contornos de los objetos que aparecen en una imagen, dándoles mayor nitidez. Lo anterior es requisito previo para realizar la segmentación.

A continuación se describen los métodos más comunes para la detección de bordes.

EL OPERADOR GRADIENTE.

El operador gradiente G aplicado sobre una imagen f(x,y) se puede definir como:

$$G[f(x,y)] = |G_x| + |G_y|$$

Ec.11 Operador Gradiente.

Siendo las derivadas parciales:

$$G_x = \frac{\partial f}{\partial x} = f(x,y) - f(x-1,y)$$

$$G_y = \frac{\partial f}{\partial y} = f(x,y) - f(x,y-1)$$

Ec. 12 Componentes del Operador Gradiente.

Estas operaciones representan la extracción de la derivada en una dirección de la pantalla y dan como resultado un valor pico cuando hay una mayor tasa de variación en el nivel de gris entre pixeles.

EL OPERADOR LAPLACIANO.

En algunos casos las transiciones de los niveles de grises son demasiado abruptas por lo anterior se prefiere tomar la segunda derivada, ya que su signo proporciona un valor que representa la dirección del cambio, es decirl si el cambio es de oscuro a claro ó de claro a oscuro. Además toma un valor de cero justo en el borde y donde se uniformizan las intensidades de gris.

La definición matemática del operador laplaciano es:

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

Ec. 13 Operador Laplaciano.

Cuando existe un cambio de oscuro a claro, la segunda derivada cambia de un valor positivo a negativo y viceversa.

2.3.18 SEGMENTACION DE LA IMAGEN.

En aplicaciones donde se requiere extraer la información de una imagen es imprescindible aplicar segmentación. Este termino se refiere a la división de la imagen en regiones que corresponden a unidades estructurales en la escena o en la distinción de objetos de interés. La segmentación es uno de los pilares básicos de la visión artificial, ya que permite resaltar un objeto separándolo del fondo, una de sus aplicaciones es reconocer si un objeto corresponde a una imagen patrón o de referencia (reconocimiento de objetos).

La segmentación se basa en tres propiedades:

a) Similitud: Cada uno de los elementos tiene propiedades parecidas de color, textura1 etc.
b) Discontinuidad: Los objetos destacan del entorno y tienen por tanto unos bordes definidos. Lo mismo ocurre si existe un ocultamiento parcial de un objeto por parte de otro. La segmentación fallará si no se a delimitado adecuadamente al objeto de interés.
c) Conectividad: Los pixeles deben de estar agrupados.

Las anteriores propiedades se realizan por medio de herramientas de procesamiento de imágenes básicas, tales como umbralización, detección de bordes, filtros digitales, etc.
Existe una gran cantidad de métodos utilizados para lograr la segmentación en una imagen, los más usados son:

SEGMENTACIÓN BASADA EN LA DETECCIÓN DE BORDES.

Este método se basa en la información suministrada por los detectores de bordes antes mencionados. La desventaja de este método es que proporciona errores cuando varios objetos están en contacto o semi ocultos por otros o por discontinuidades en los bordes a causa del ruido. Para solventar dicha dificultad se aplican otros métodos que analizando los puntos detectados determinen las fronteras entre objetos.

EL GRADIENTE.

El método del gradiente se basa en las primeras y segundas derivadas. Primero se saca el módulo y el argumento del gradiente, luego se agrupan los pixeles pertenecientes a los bordes con sus vecinos. Para que dos pixeles pertenezcan al mismo borde deben cumplir que:

- Estar conectados.
- Y que la diferencia entre gradientes y ángulos sea menor que un valor prefijado:

$$|g_1 - g_2| < T_g$$

$$|\theta_1 - \theta_2| < T_\theta$$

Ec. 14 Condiciones límites para el Gradiente.

LA TRANSFORMADA DE HOUGH.

El método descrito anteriormente tiene en cuenta solo la vecindad de cada pixel, en ocasiones es necesario basarse en la información de la imagen completa, siendo la transformada de Hough el método indicado.

La ecuación de una línea recta puede escribirse como:

$$x\cos\theta + ysen\theta = \lambda$$

Ec. 15 Ecuacion de la línea recta parametrizada con θ.

Quedando por tanto cada recta definida por su par de parámetros (ρ,θ).

Todos los puntos que pertenezcan a la misma recta coincidirán en las componentes ρ y θ. Se buscan por tanto todas las posibles rectas que unan los puntos detectados como bordes y se obtienen sus valores ρ y θ. Al final aquellos valores más repetidos coinciden con la recta que contiene el mayor número de pixeles.

SEGMENTACIÓN DE REGIONES.

La segmentación por regiones se basa en el principio de que los objetos presentan una uniformidad en alguna o algunas características. Se buscan por tanto aquellos pixeles que estando unidos presentan una propiedad común.

2.3.19 RECONOCIMIENTOS DE OBJETOS

En muchas aplicaciones es necesario determinar los objetos presentes en una imagen (detección de etiquetas, de señales de transito, etc.), para controles industriales, análisis médicos, medios publicitarios, etc. Este proceso es una de las últimas etapas dentro de lo que es la visión artificial y una de las más difíciles de llevar a cabo.

El reconocimiento de objetos consiste en buscar un objeto y compararlo con un patrón ya dado (imagen predefinida), y toma una decisión si al compararlos son iguales o diferentes.

Las etapas para lograr el reconocimiento de objetos son:

1. Adquirir la imagen y darle las características optimas para la aplicación que se desea. (aplicar filtrado de ruido, umbralización, etc.)
2. Segmentar la imagen a procesar, es decir obtener todos los objetos presente en la imagen. (formación de hipótesis).
3. En base a la hipótesis se comparan los objetos encontrados en la imagen con el patrón (disponible en una base de datos). Si encontró el objeto la búsqueda acaba, sino, continua la iteración hasta terminar de comparar todos los objetos encontrados en la imagen.

CAPITULO III.

FUNDAMENTACION EXPERIMENTAL PARA EL DESARROLLO DEL PROYECTO.

Debido a la naturaleza del proyecto y a los componentes usados en el desarrollo del mismo, fue necesario realizar múltiples experimentos para conocer el comportamiento real de los elementos funcionando en el ambiente a utilizar. Los elementos con los cuales se experimentó son:

- ***La cámara***; en la cual es necesario conocer la sensiblidad a la luz, su campo visual y su distancia a la escena (entorno controlado).
- ***La tarjeta digitalizadora***, en donde es necesario determinar la resolución, controles de contraste, brillo, y la velocidad de la digitalización; cuyo comportamiento depende de la rapidez de la computadora.
- ***Puerto serie***, a través del cual se controla el brazo robot y en él que se necesita conocer como programarlo desde Visual C++.
- ***El brazo robot***, en el cual se necesitan determinar sus distancias de alcance máximo, velocidad de sus movimientos, resolución y precisión angular.
- ***Entorno controlado***, en el cual se incluye el fondo y los objetos de la escena, en donde interesa determinar el color, la forma y comportamiento de la luz sobre ellos.
- ***Lenguaje de Programación (Visual C++)***, el cual es responsable del procesamiento de imágenes y del control del brazo robot; en donde es necesario aprender a crear librerías, archivos DLL's, entornos gráficos, control de hardware y manejo de grandes bloques de memoria.

3.1 EXPERIMENTOS CON LA CAMARA.

Se experimentaron con dos tipos de cámaras.

a) Cámara tipo Vidicom, la cual dió buenos resultados en ambientes no muy brillantes, presentaba efecto de cola de cometa y mostraba baja sensibilidad a la intensidad de la luz. Necesitaba demasiada luz ambiental para mostrar un contraste y brillo adecuados.

b) Cámara CCD, presentó la desventaja de ser muy sensible a la luz, las ventajas de tener muy poco ruido y ser compacta. Esta fue seleccionada para el proyecto (ver características en anexo A) debido a su alta tolerancia al ruido, eliminación del efecto de "lag" o retardo y su facilidad de transporte por ser más compacta y liviana. Esta cámara presenta la mejor nitidez de imagen. Su costo es mayor que el de la cámara Vidicom, pero esto es compensado por sus ventajas.

Además se experimentó con diferentes tipos de lentes:

1. Teleobjetivo: Necesitaba demasiada distancia para captar más espacio de la escena (menor ángulo visual), posee un iris que permite la regulación de la luz que incide sobre el elemento sensor de la cámara.

2. AutoZOOM: era necesaria una señal eléctrica extra para controlar el ZOOM, aunque para futuros proyectos puede ser una ventaja tenerlo para evaluar regiones específicas de la escena.

3. Panorámico: Es un lente sencillo, el cual permite tener un mayor ángulo de visión , menor distancia entre la cámara y la escena, pero no posee un iris variable que pueda ser útil para compensar la alta sensibilidad de la camara CCD. Este último fue seleccionado para el proyecto debido al mayor ángulo de visión a muy poca distancia.

Es importante mencionar que la cámara se encuentra con un ángulo de inclinación (determinado experimentalmente), esto ayuda a determinar profundidad en el entorno y en los objetos.

3.2 EXPERIMENTOS CON LA TARJETA DIGITALIZADORA.

La tarjeta fue comprada en el extranjero, el modelo es PIXCI-SV4 la cual necesita que el bus PCI de la PC pueda ser configurado (normalmente en el BIOS SETUP) en modo BURST (Ver anexo B).

En un inicio se conectó a una computadora Pentium 133MHz / 16Mbyte de RAM dando como resultado una lentitud en la digitalización de las imágenes y, al sobrecargarse la memoria, la PC se bloqueaba. Por lo tanto fue necesaria la adquisición de una nueva PC , que es la utilizada actualmente en el proyecto, teniendo las siguientes características:

- Bursting Mode PCI Bus
- Bus PCI a 100MHz
- Pentium II.350MHz
- 32MB de RAM.

3.3 EXPERIMENTOS CON EL PUERTO SERIE.

Fue necesario crear un archivo DLL para el control del puerto serie, ya que se debe configurar con las siguientes características: 8 bits de datos, 1 stop bit, sin paridad y a 9600 baudios; de acuerdo a los requerimientos del brazo robot.

El archivo DLL se basa en la instrucción outp (perteneciente a la librería CONIO.H), que realiza escrituras a las direcciones de los registros de la UART PC 16550.

Para realizar dicha configuración se debe conocer la dirección del puerto serie, por ejemplo COM1=03F8 (el caso del proyecto). Es necesario ver el anexo C como referencia de los registros de la UART .

De acuerdo a la tabla II (Anexo C) el mapeo para direccionar a los registros queda de la siguiente manera:

03F8---->Registro 0
03F9---->Registro 1
03FA---->Registro 2
03FB---->Registro 3

03FC---->Registro 4
03FD---->Registro 5
03FE---->Registro 6
03FF----->Registro 7

Se debe tener muy en cuenta el bit 7 del registro 3 (DLAB), el cual determina el uso del registro 0 y 1.

3.4 EXPERIMENTOS CON BRAZO ROBOT.

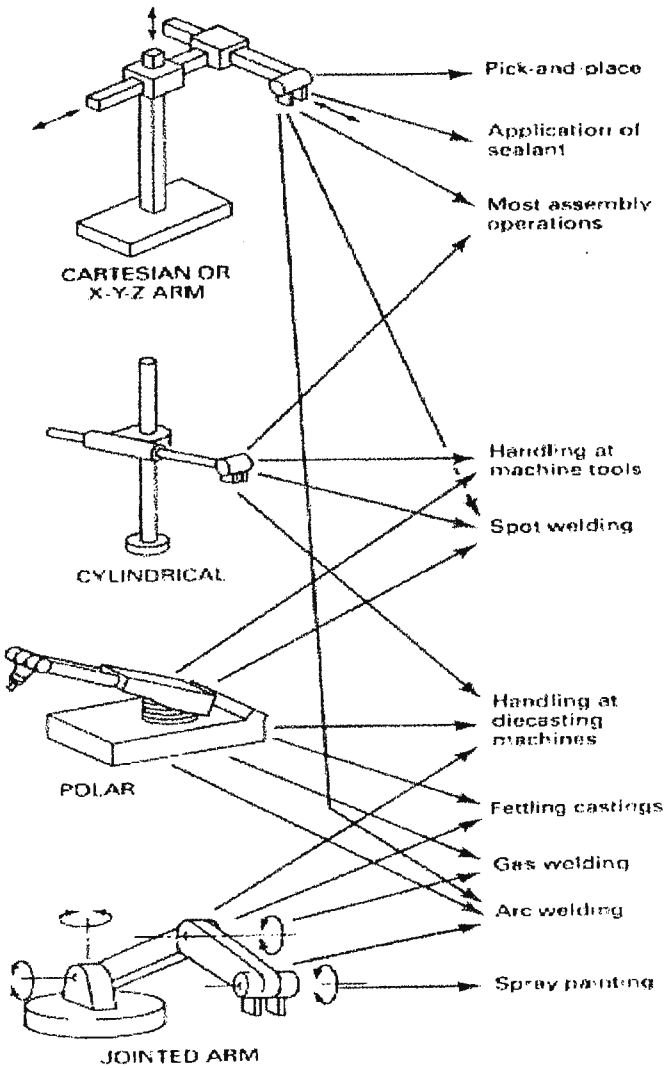


Fig. 31. Diversos tipos de brazos, construidos según la aplicación.

Es importante mencionar que existen diferentes tipos de brazos robot de acuerdo a la aplicación y al tipo de coordenada a utilizar, la figura anterior ilustra los tipos de brazos para las coordenadas mas comúnmente usadas. El brazo utilizado en la presente tesis es el denominado JOINTED ARM el cual tiene la característica de poseer la mayor flexibilidad en los tipos de movimientos, pero no se apega a ningún sistema de coordenadas específico, dando como resultado un mayor grado de dificultad en su control.

El brazo robot fue comprado a través de Internet (www.lynxmotion.com), siendo necesario ensamblar sus piezas de acuerdo a las instrucciones que aparecen en el anexo E. Al momento de ensamblarlo se determinaron las posiciones de inicio, las cuales tomarán efecto al aplicarle energía eléctrica al brazo. El brazo usado se muestra a continuación:

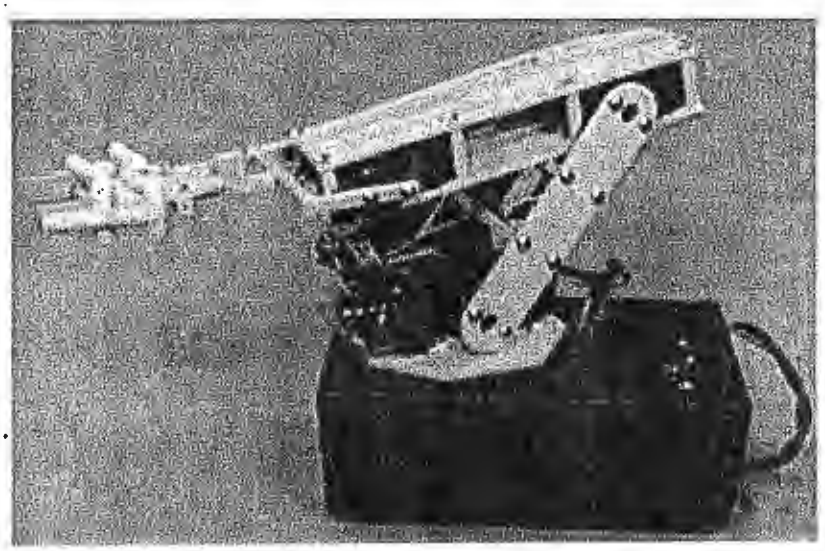


Fig. 32 Brazo robot utilizado en el proyecto, aspecto al estar recién ensamblado.

El brazo es movido usando servo motores y estos controlados a través del periodo de un pulso digital (Ver anexo E). El kit incorpora una tarjeta que convierte los valores de un byte (0-255) en el ancho del pulso que controla a los motores. La tarjeta necesita tres bytes (usando comunicación serie), el primero es una constante (255) que indica "wake up", el segundo es un número de 0 a 7 el cual es el indicador del motor a mover, y por último el valor hasta el cual se

debe mover el motor (0 bytes--->0 grados, 255 bytes --->90 grados). Experimentalmente se obtuvo que por cada aumento o decremento de este último valor se tiene un cambio de 0.418 grados y no 0.36 grados como dice la hoja técnica de los motores. También se obtuvieron los siguientes valores para posiciones claves de cada parte del brazo:

Fig 33.

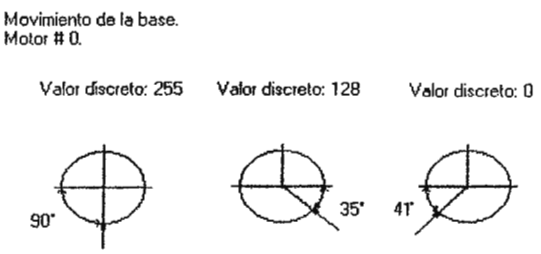


Fig. 34.

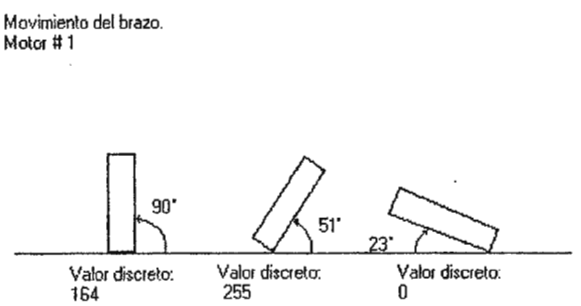


Fig. 35.

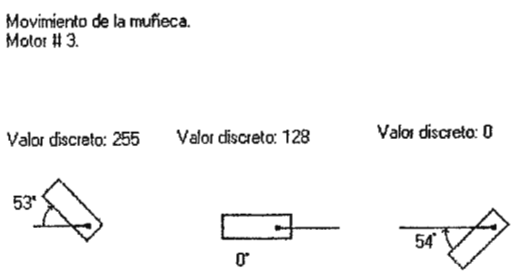
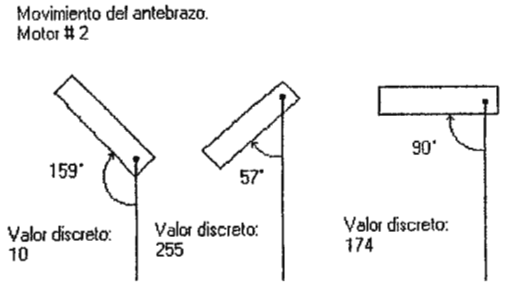


Fig. 36.



Para darle naturalidad al movimiento del brazo, se va gradualmente enviando valores sucesivos a cada motor para irlo acercando a la posición de manera progresiva. Entre cada envío de datos de una nueva posición se coloca un retardo, que de no existir provocaría una saturación de información en la tarjeta controladora del brazo.

Es importante señalar que al momento de mover el brazo se debe de recordar que éste tiene una inercia, por lo cual se evitan los movimientos bruscos. Esto se consigue utilizando el retardo mencionado.

El posicionamiento del brazo tiene la interesante característica de ser tridimensional pero regido por una referencia (la imagen) bidimensional. De lo anterior se desprende que en todo caso la ubicación por medio de una imagen será siempre una aproximación, pues cada pixel representa una infinidad de puntos del mundo espacial. Dicho de otra forma las coordenadas tridimensionales determinadas por los valores de posición de pixeles, siempre tendrán un margen de error debido a la naturaleza de la situación.

Se disponía de tres alternativas para controlar el movimiento total del brazo, éstas son:

1- Ecuación Vectorial, en este caso se busca una sola ecuación que incluya el movimiento de los 6 motores como un todo. Aqui interesa manejar proyecciones y ángulos resultantes.

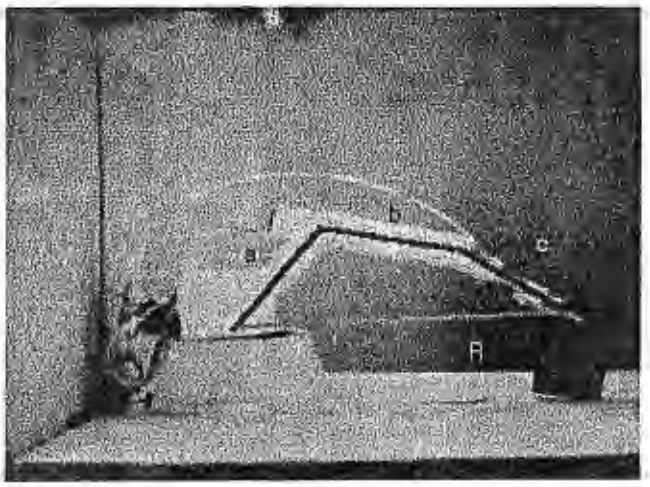


Fig. 37. Representación gráfica de los vectores necesarios para determinar el movimiento del brazo.

En la figura se muestran los vectores que corresponden a cada segmento del brazo (segmentos "a", "b" y "c") y el vector resultante (segmento R) el cual es variable en longitud y dirección. La

idea es obtener una ecuación que involucre los cuatro ángulos internos de los vértices del polígono formado por los vectores antes mencionados. Se debe tomar en cuenta que la longitud de los vectores son de longitud constante (excepto el resultante) y sólo varía la dirección de acuerdo a la posición del brazo.

*2- **Tabla de valores***, se construye una matriz con valores discretos determinados experimentalmente, construyendo un conjunto de arreglos que se manejan como matrices. A partir de estos valores se busca una combinación de valores que correspondan a la posición más próxima a la requerida. Este procedimiento tiene la desventaja de ser demasiado aproximado por trabajar con valores fijos.

*3- **Ecuación Polinómica***, cada uno de los motores se mueve en base a una relación matemática de orden mayor a uno. Se miden valores para cada motor en posiciones deseadas y se plotean los puntos. Para obtener la expresión de posicionamiento se usa el método de los mínimos cuadrados. Este procedimiento se usa en el proyecto y para mayores detalles ver el capítulo 4 y el anexo F.

3.5 EXPERIMENTACION PARA LA DETERMINACION DE LAS CARACTERISTICAS DEL ENTORNO.

3.5.1 FORMA DEL ENTORNO.

La primera forma del entorno se muestra en la figura siguiente:

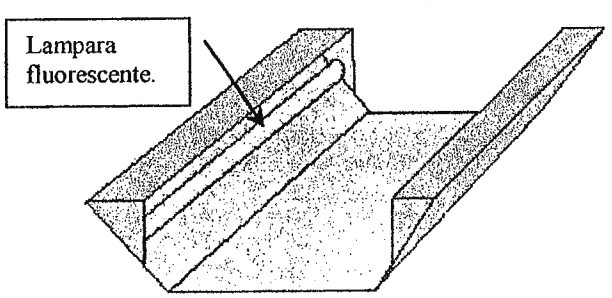


Fig. 38 Primer entorno.

La cual presentó la desventaja de no permitir reubicar la iluminación para acomodarla al ambiente. Además, las lámparas se encontraban muy cerca de los objetos (de su base) y por la sensibilidad a la luz de la cámara CCD se suprimían detalles debido a los reflejos.

La forma final que tiene el entorno es:

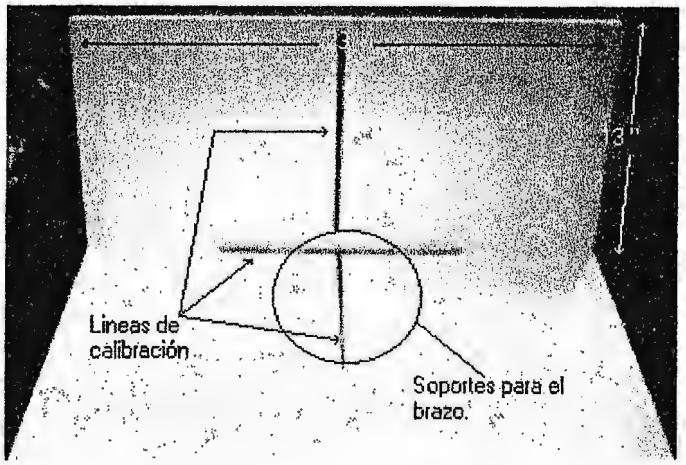


Fig. 39 Aspecto final del entorno. Dimensiones de altura y ancho.

La imagen anterior presenta la proyección que es usada durante el proyecto, es decir, la cámara de video se coloca frente al entorno y al brazo.

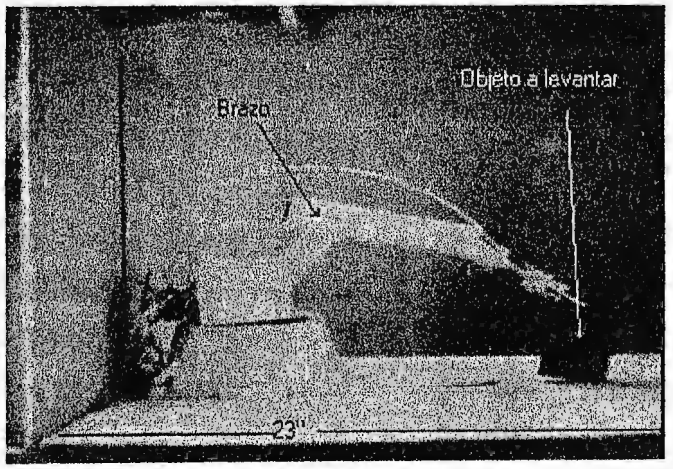


Fig. 40 Entorno con el brazo robot en acción. Vista de perfil.

Las imágenes de la Fig.36 y 37 fueron tomadas sin usar el sistema de iluminación. Las lámparas utilizadas son de luz blanca (fluorescentes), éstas tienen reflectores parabólicos y presentan la ventaja de tener facilidad para reubicarse, ya que poseen un sistema mecánico tipo brazo para posicionarlas de diferentes maneras, sin que éstas deban estar sujetas a las estructuras del entorno.

3.5.2 COLOR DEL ENTORNO.

Al inicio, se experimentó con color negro de fondo y objetos en blanco, dando los siguientes resultados:

- Las sombras de los objetos son eliminadas pues no hay sombras sobre negro.
- Al usar superficies negras pulimentadas (formica negra) se producen reflejos y al ser rugosas (lona negra) no hay reflejos.
- Al ser blanco el objeto refleja tanta luz que se pierden sus bordes internos.
- Hay una mayor sensibilidad al ruido al ser fondo negro.
- El rango para umbralizar es crítico.

El color usado en el proyecto para el fondo del entorno es blanco y los objetos son negros, presentando las siguientes características:

- Las variaciones de iluminación no son críticas ya que siempre el blanco producirá un reflejo de cualquier intensidad de luz.
- Debido a lo anterior el rango para umbralizar no es crítico.
- Como los objetos son negros se resaltan más los bordes externos.
- Presenta poca sensibilidad al ruido.
- Tiene la desventaja de que la sombra es muy notoria, pero este efecto se reduce aplicando luz distribuida.
- Los objetos transparentes no se pueden detectar.

3.5.3 UBICACIÓN Y COLOR DEL BRAZO ROBOT.

La ubicación del brazo fue definida como lo muestra la siguiente figura:

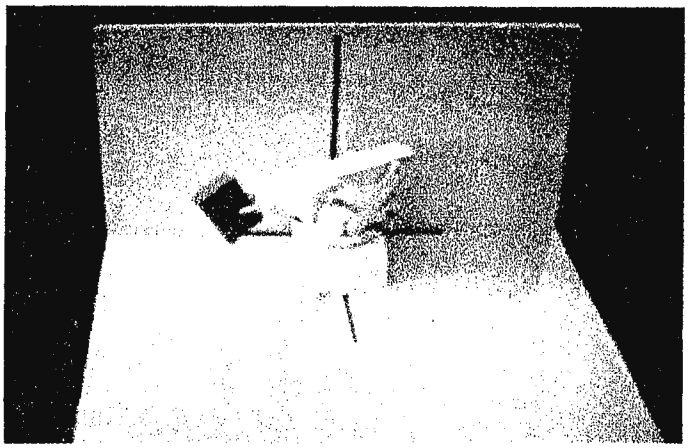


Fig. 41 El brazo adaptado a las características del entorno.

Esta posición evita que el brazo llegue en un momento dado a colocarse sobre los objetos.
Ya que el brazo esta dentro de la escena (lo cual es desventaja para el procesamiento de las imágenes), se pintó del mismo color que el fondo (blanco) y de esta forma se volvió indetectable para la cámara.

3.6 EXPERIMENTACION CON VISUAL C++.

Se tenían conocimientos de programación en Turbo Pascal y en Borland C, pero se desconocía la programación y uso de Visual C++, por lo cual, estos experimentos fueron los primeros en llevarse a cabo. Dentro de los estudios y experimentos llevados a cabo están:

- Familiarización del entorno Visual C++.
- Uso del AppWizard y Class Wizard.
- Creación de clases.
- Creación y uso de archivos DLL's.

- Incrustación de ventanas gráficas.
- Incrustación de objetos (botones, cajas de lista, cajas de edición, mensajes de alerta,etc).
- Manejo del puerto serie.
- Manejo de buffers para manipulación de video.

Es de hacer notar que se dispone de dos maneras de programar y cada una presenta sus respectivas ventajas, éstas son: *Programación estructurada clásica* y *Programación utilizando objetos*.

En este proyecto era necesario realizar los algoritmos de procesamiento de imágenes por medio de funciones ya que se necesita en diversos casos realizar la misma operación en diferentes puntos del programa o se realizan procesos repetitivos.

La programación objeto presenta la gran ventaja de ser la natural de Visual C++ y con la que se ha creado Windows 3.1 / 95 / 98. Para esto, Visual C++ tiene un creador de esqueletos de aplicaciones que se denomina *App Wizard* que permite generar una aplicación totalmente compatible con Windows y de tipo visual.

Una vez creado este esqueleto se procede a crear el código con el cual cada icono o herramienta visual creada funcionará.

Una vez se tiene creado el esqueleto se puede utilizar *Class Wizard* (Ctrl + W) para poder ligar por medio de programación objeto cada elemento a usar en el programa con la parte visual de la aplicación. Usando *Class Wizard* se pueden ligar variables miembros de cada objeto (denominado también clase) y adicionar funciones que ejecuten eventos dentro del programa. Estos eventos son acciones con el teclado, con el mouse o la inclusión de "rutinas" pertenecientes a Windows que se denominan MFCs.

CAPITULO IV

ALGORITMOS DESARROLLADOS.

4.1 PRELIMINARES.

Antes de introducirse al texto se deben de tomar en cuenta los siguientes detalles:

- Para procesar cada una de las figuras se requiere de gran cantidad de memoria (bloques de RAM) que no tengan conflicto y que sean de rápido acceso para agilizar el programa. Esto se consigue manejando punteros para procesar las imágenes y usando las funciones de *Malloc* para asignarles espacios de memoria que no creen conflictos. Un puntero es semejante a un arreglo de variables pero en las cuales se direcciona directamente a memoria para el acceso del dato (esto le da rapidez al procesamiento de la información). Para mayores detalles referirse a un manual de Visual C++. Se puede tener más de un buffer activo al mismo tiempo, esto es muy dependiente de la cantidad de memoria del sistema. Para el caso de este proyecto el máximo de buffers que pueden ser abiertos al mismo tiempo son 20.
- La resolución de una imagen tiene relación directa con el tamaño del bloque de RAM que se requiere manejar. Una resolución baja da agilidad al programa (por tener menos datos a procesar), pero presenta el incoveniente de que el espacio continuo del mundo real (presentado en la imagen) se vuelve más discreto. Una resolución alta permite definir muchos puntos de distancia dentro de la imagen (con respecto al mundo real) lo cual da una mejor aproximación al espacio continuo.
- Se usa la función outp (de la librería CONIO.H) de Visual C++, para poder programar directamente la interfaz serie de la computadora. Esto se requiere debido a que el brazo robot es controlado por medio del puerto serie. Para referencias acerca del dispositivo que controla la comunicación serie en la PC usada, remítase a los anexos.
- Un pixel queda completamente definido por sus coordenadas "x" (columna), "y" (fila) y por su valor de brillantez (de 0 a 255).
- La coordenada 0,0 de un archivo BMP de imagen se encuentra en la esquina superior izquierda y las coordenadas máximas se encuentran en la esquina inferior derecha de la

misma. El valor de estas coordenadas extremas depende de la resolución asignada a la imagen.

- Todos los algoritmos, de procesamiento de imagen, asumen que la imagen tiene solamente dos valores: negro y blanco, este último identifica al fondo.

4.2 ALGORITMO PARA FILTRO DE LA MEDIANA.

Como se mencionó anteriormente se usa este filtro para realizar un acondicionado de la imagen que no tenga ruido y que no pierda los rasgos (en forma notable) de los contornos de las figuras. Para realizar este filtro se usó dos buffers de memoria, uno para tener a la imagen original y otro para contener a la imagen que recibe el filtrado, con este último buffer se sigue el procesamiento de las imágenes.

Primeramente se toma un vecindario de 3x3. En este vecindario el pixel central (cuyo valor se va a modificar) tiene coordenadas "x", "y" (donde la primera es el valor de columna y la segunda es el valor de fila, comenzando en la esquina superior izquierda de la imagen); por lo que el vecindario que se usa tiene estas coordenadas:

x-1, y-1	x, y-1	x+1, y-1
x-1, y	x, y	x+1, y
x-1, y+1	x, y+1	x+1, y+1

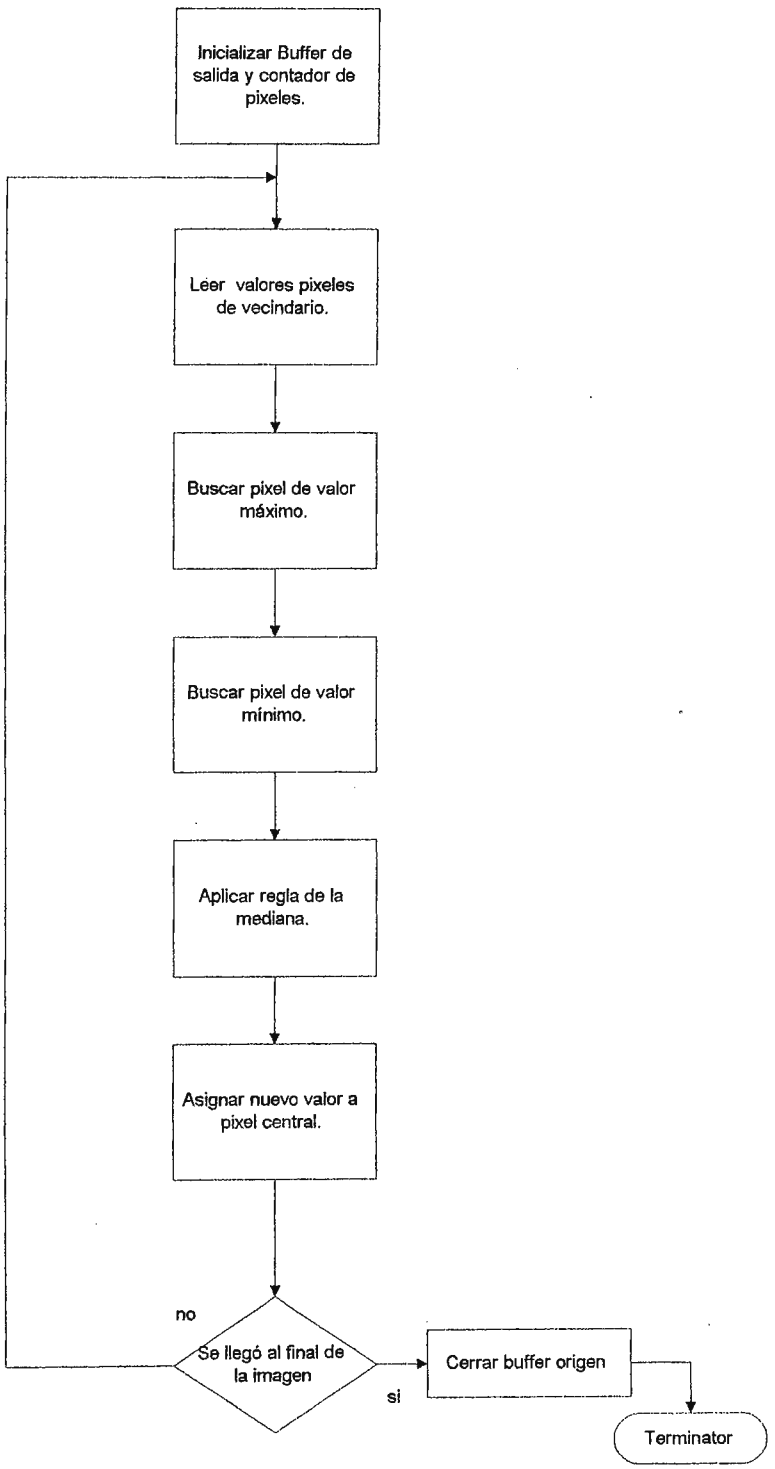
Para controlar estos valores se usa un par de lazos (For en el caso de Visual C++), se busca el valor máximo y el valor mínimo presente en los pixeles de este vecindario, se almacenan en variables separadas y se le aplica la siguiente regla:

Valor de pixel central = (máximo + mínimo) / 2

Ec. 16 Valor asigando al pixel central en el filtro de la mediana.

Esto se realiza para la imagen entera, por lo que al final la mayoría de los pixeles han sido cambiados en base a la regla anterior. Para hacer el algoritmo más eficiente, se usan en el cálculo los valores de la imagen original y no participan los pixeles cambiados por la operación, pues podría generarse un error acumulativo.

ALGORITMO PARA APLICACIÓN DE FILTRO DE LA MEDIANA.



4.3 ALGORITMO PARA EL UMBRALIZADO.

Para separar a las figuras del fondo se requiere de un proceso de doble umbralizado. El fondo será blanco y las figuras negras (con valores extremos que son realmente una binarización). Las figuras son negras en la realidad, por lo que tienen un valor de brillantez muy cercano a 0 y el fondo que es muy cercano a blanco presenta una brillantez cercana a 255.

Se evalúa si el valor se encuentra entre 0 y un umbral máximo de gris (que puede ser modificado) y a los valores que se encuentren entre ese intervalo se colocan a cero. De no cumplir con la regla (es decir, el complemento de la gama de grises del intervalo anterior) se coloca a 255, que equivale a blanco.

Una demostración del concepto anterior se muestra en la siguiente imagen:

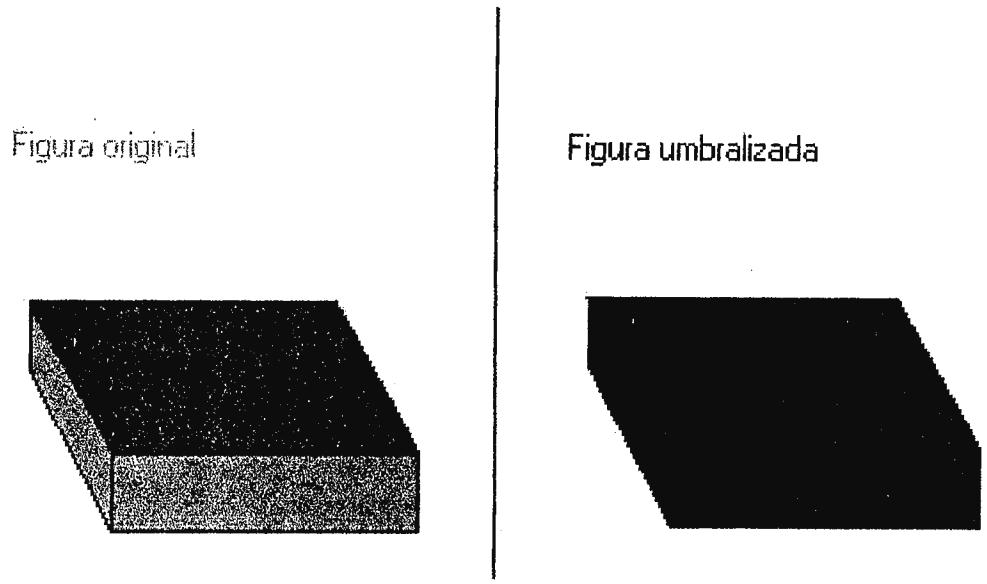


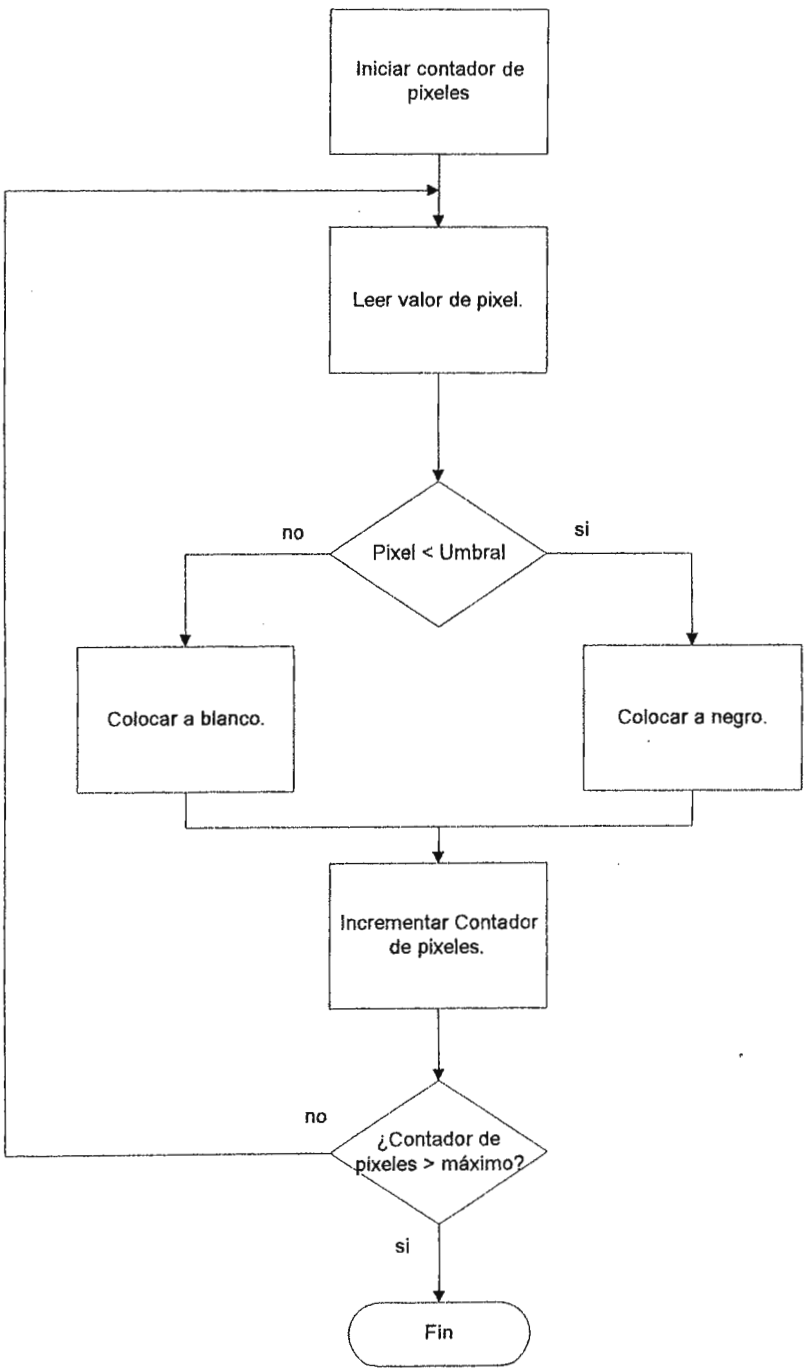
Fig. 42 Figura antes y después del umbralizado.

En la imagen anterior se ha hecho el efecto de exagerar los pixeles para proporcionar una idea de la nueva apariencia de los objetos después de que se le han reasignado sus valores de pixeles.

El proceso finaliza cuando se recorre la imagen entera. Es importante notar que en la práctica se encuentra que el valor de umbral máximo de las figuras es muy dependiente de la iluminación presente en la toma de la imagen. Para disminuir la dependencia de la iluminación se puede adicionar un valor (o restar) a todos los pixeles de la imagen, con lo que se cambia el valor de la brillantez total de la imagen. Este algoritmo equivale a modificar la brillantez de la imagen.

En este proceso se necesita solamente de un buffer para manipular la imagen.

ALGORITMO PARA EL UMBRALIZADO DE LOS OBJETOS Y EL FONDO.



4.4 ALGORITMO PARA LA MEDICION DEL AREA DE LAS IMÁGENES, SU CENTRO GEOMETRICO Y EL CONTEO.

Estos tres procesos, aunque son separados desde el punto lógico, se diseñaron en una sola porción de programa.

Para ello se parte de que las figuras son negras y el fondo es blanco.

Se usa una estructura de datos (es una variable que engloba a muchas otras de igual o diverso tipo) en la cual se guardará el valor del área (que es simplemente un conteo de pixeles), se guardarán las esquinas extremas que encuadran a la imagen y se colocará el valor (x ,y) del centro geométrico.

Se inicia una variable contadora de figuras a cero y se define que la máxima cantidad de figuras a presentarse en el entorno será de 20 (esto es necesario debido a que se usa un arreglo de estructuras, una estructura por figura).

Los pixeles se leen de izquierda a derecha y por filas. La lectura de los valores de pixel se hace en parejas contiguas que se encuentran en la misma fila, pues no se busca un valor de pixel sino más bien un valor de cambio.

El programa inicia buscando un valor de cambio de un pixel blanco a la izquierda y un pixel negro a la derecha (que equivale a buscar el contorno izquierdo de todas las figuras). Cuando se encuentre esta transición la primera vez se aumenta el contador de figuras en uno (pues ya se tiene una primer figura) y se guarda esa esquina (de la transición) como la esquina superior izquierda de un rectángulo que encuadra exactamente a la figura.

Al seguirse moviendo en la misma fila se leen dos valores contiguos de negro (pues se encuentra dentro de la figura umbralizada), por lo que ahora el programa busca un par de valores en los que el pixel izquierdo sea negro y el derecho sea blanco (esto corresponde al contorno derecho de la figura). Al encontrar esta transición se sabe que la figura ha terminado y se baja una fila para continuar analizando la figura.

Al llegar al contorno derecho de la figura este par de coordenadas (x, y) se guardan como la esquina inferior derecha del rectángulo que enmarca a la figura.

Mientras se encuentra la lectura de pixeles dentro de la figura (el par de valores contiguos con valores de 0) se realiza un proceso destructivo de colocar ese par de valores a 255 (blanco como el fondo). Esto se realiza con el propósito de que cada figura contada sea eliminada de la imagen, para que no interfiera con el resto de las figuras cuando sea el momento de analizarlas.

Simultáneamente a la eliminación de los pixeles negros se va incrementando el contador de área en uno por cada pixel que se lee. Mientras este contador aumenta, se va realizando la suma de las posiciones correspondientes a "x" y "y" del objeto en consideración.

Al bajar en fila se repite el proceso anterior de búsqueda de la transición de blanco a negro y se hacen los mismos pasos previos. Es importante señalar que al momento de analizar cada fila de la figura se vuelven a calcular las coordenadas de la esquina superior izquierda y la esquina inferior derecha del rectángulo que enmarca a la figura bajo análisis. Cada vez que se encuentra un cambio de blanco a negro o viceversa se comparan sus coordenadas (x, y) (columna, fila) para saber si se trata de una figura diferente o de la misma. Al ser de la misma figura las variaciones con el valor previo de la fila anterior son muy cercanos, mientras que si se tratase de una figura diferente los valores tendrían una diferencia grande.

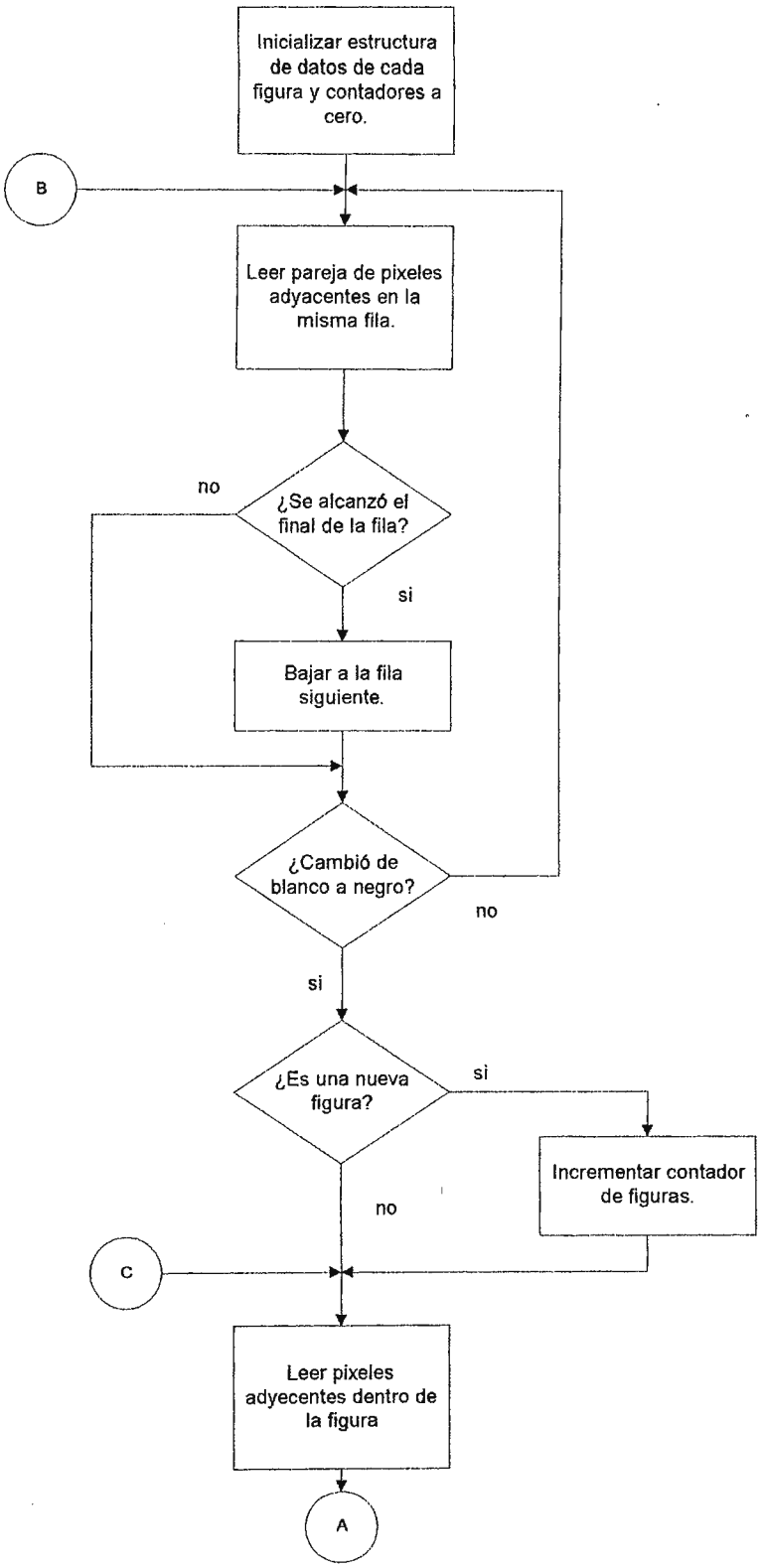
Se sabe que acabó una figura cuando ya no hay más líneas de negros que leer. En este caso la figura entera habrá sido borrada, y se dispone de la sumatoria de todos sus pixeles y de las esquinas superior izquierda e inferior derecha del rectángulo que la enmarca. Se toman los valores de las sumatorias de las coordenadas en "x" de todos los pixeles y se divide entre el número total de pixeles (área) para obtener el "x" del centro geométrico, lo mismo se hace con "y".

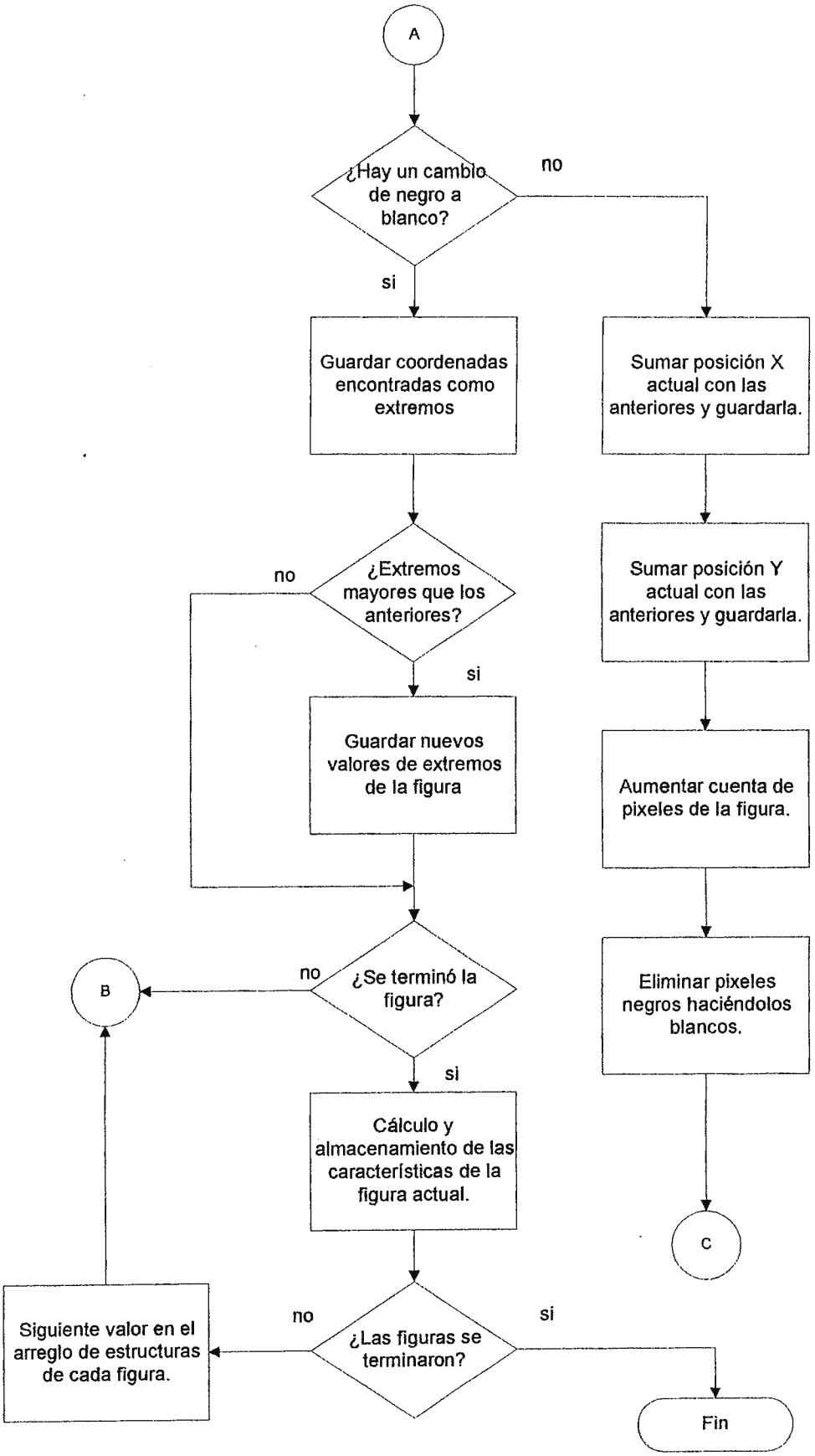
Terminada de analizar esta primer figura se procede a buscar un nuevo objeto, y de encontrarse se repiten los pasos anteriores. Esta búsqueda de nuevas figuras se hace leyendo el buffer desde el inicio. Como debe de suponerse, al encontrar otro objeto el contador de figuras se incrementa en uno.

¿Cómo "sabe" el programa que ya no hay más figuras? Debido a que cada vez que se procesa un pixel (negro) de la figura se le borra (se coloca a blanco), el programa detecta que ya no hay más figuras cuando haga un recorrido entero por el buffer (llegue a las coordenadas de columna y fila extremas) y todos los pixeles sean blancos, o mejor dicho, sin ninguna transición de blanco a negro.

De lo anterior se desprende que el proceso anterior es destructivo, pues al final de todo ya no hay figuras sino solamente fondo. Esto hace necesario que antes de correr esta rutina se deba hacer una copia del buffer original.

ALGORITMO PARA CONTEO DE FIGURAS, EXTRACCION DE AREAS, CENTROIDE Y COORDENADAS LIMITE DE CADA OBJETO.




A
¿Hay un cambio de negro a blanco?
no
si
Guardar coordenadas encontradas como extremos
Sumar posición X actual con las anteriores y guardarla.
¿Extremos mayores que los anteriores?
no
si
Sumar posición Y actual con las anteriores y guardarla.
Guardar nuevos valores de extremos de la figura
Aumentar cuenta de pixeles de la figura.
B
¿Se terminó la figura?
no
si
Eliminar pixeles negros haciéndolos blancos.
Cálculo y almacenamiento de las características de la figura actual.
C
Siguiente valor en el arreglo de estructuras de cada figura.
¿Las figuras se terminaron?
no
si
Fin

4.5 DETERMINACION DE LAS COORDENADAS CARTESIANAS DE LOS OBJETOS.

Se dispone de los centros geométricos y coordenadas del rectángulo que enmarca a las figuras. Esto lo proporciona el algoritmo anterior. Los datos a usar por este algoritmo se muestran en la figura siguiente:

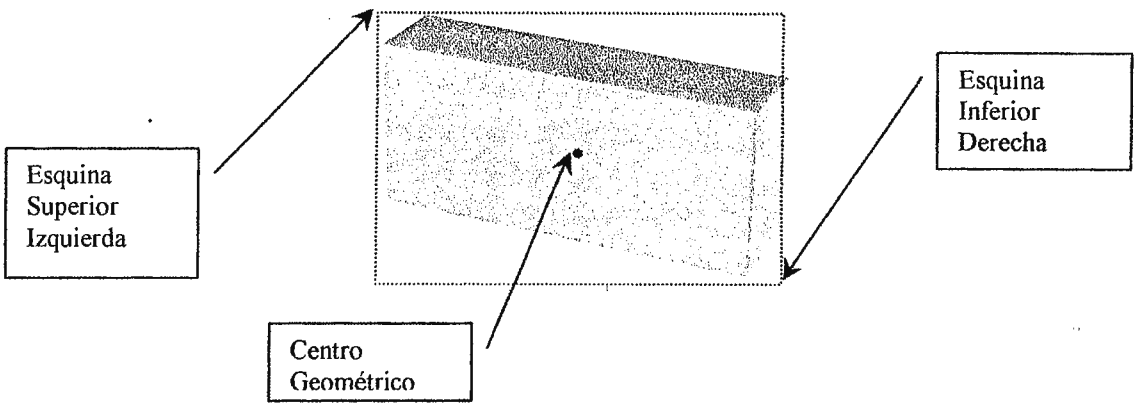


Fig. 43 Objeto con todas las coordenadas de pixel que se le calculan.

Los siguientes algoritmos se basan en el trazado de cuadrículas en un buffer previamente preparado; solamente Z no, pues se obtiene a partir de Y. Estas cuadrículas responden a ecuaciones obtenidas experimentalmente por medio del método de mínimos cuadrados.

Antes de la determinación de cada una de las coordenadas se realiza una selección de una de las figuras, ya sea la mayor, la menor, o la que cumplan con algún rango de tamaño o de posición de centroide en pixeles (por ubicación dentro de la imagen). Se puede usar el algoritmo para todos los objetos, pero aumenta significativamente el tiempo de ejecución. Las direcciones de las coordenadas usadas son :

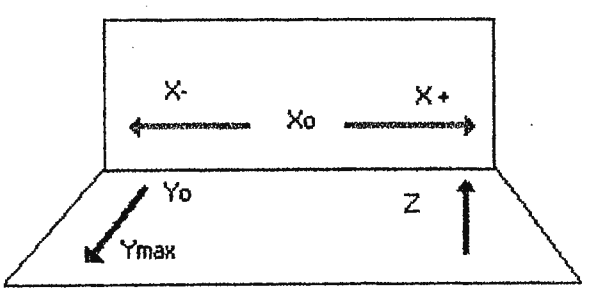


Fig. 44 Diagrama que muestra las direcciones en que se toman las coordenadas X, Y y Z dentro del entorno.

4.5.1 COORDENADA X.

El algoritmo para la coordenada X utiliza líneas trazadas en un buffer en blanco. Las líneas presentan una distorsión trapezoidal debido a la perspectiva. Las líneas se encuentran separadas una distancia de un 1 cm (con respecto al entorno) pero éstas parecen divergir debido al efecto antes mencionado. Para mayores detalles sobre la perspectiva consultar el capítulo 2, en la sección 2.3.3.

Una representación gráfica (que no incluye las líneas usadas en realidad sino una simulación) de lo anterior se muestra a continuación:

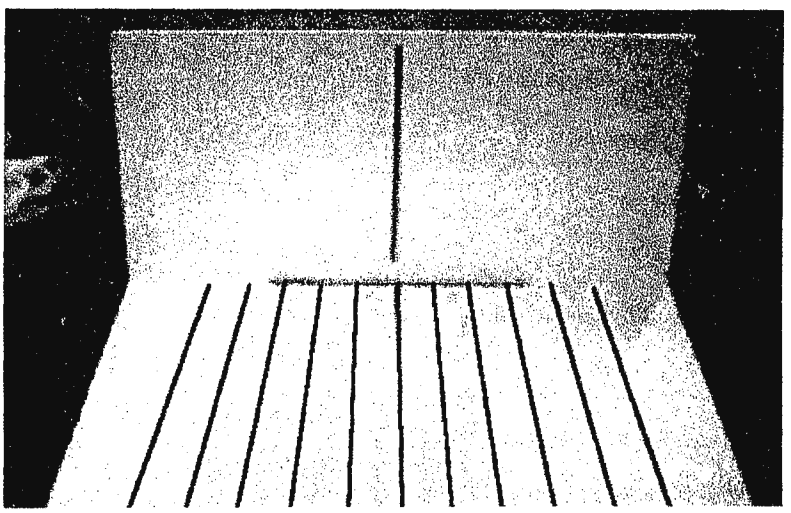


Fig. 45 Líneas que representan los cambios en la dimensión X dentro del entorno.

El algoritmo comienza preparando al buffer con sus líneas generadas a través de una ecuación punto-pendiente, las cuales buscan unir puntos obtenidos experimentalmente. Una vez determinada la figura de interés, en base al criterio de selección antes mencionado, se toma la coordenada "y" (fila) del centroide de la figura y se comienzan a leer los pixeles de esa fila, se evalúa si el pixel es negro. Encontrar un pixel en negro significa "tocar" una línea, por lo cual si eso ocurre se verifica si la coordenada actual leída no ha superado a la "x" del centroide. De no ser así se incrementa el contador de líneas X (previamente inicializado a cero), que lleva la cuenta de los centímetros; recuerdese que las líneas se han trazado para que la distancia entre ellas corresponda a 1cm. del mundo real. Después de esto, se repite el ciclo anterior.

Una vez alcanzado o sobrepasado el valor X del centroide se procede a aplicar la siguiente fórmula:

Px = (centroide X – coord. linea anterior) / (coord. linea siguiente – coord. linea anterior)

Ec. 17 Proporción menor a 1cm. tomado en coordenada X.

La cual tiene como objetivo determinar una proporción que brinde los decimales correspondientes al caso en que se tenga el "x" del centroide entre dos líneas (lo cual es el caso general). A este valor le llamaremos Proporción en X (Px)

Finalmente el valor de X se obtiene de la siguiente suma:

$$\mathbf{X = Contador\ de\ líneas\ en\ X + Proporción\ en\ X}$$

Ec. 18 Valor final de la posición en X.

Luego se le resta un valor constante para que se tenga como referencia el centro de la imagen.

4.5.2 COORDENADA Y.

Se coloca un buffer en blanco en el cual se trazan líneas horizontales de acuerdo a la siguiente ecuación (empiezan las líneas desde el pixel 107):

$$\mathbf{Pixel\ de\ fila = 0.08115N^2 + 4.42833N + 107}$$

Ec. 19 Ecuación para ubicar las líneas horizontales de la cuadrícula para coordenada Y.

Donde:

- N es el número de líneas a trazar comenzando desde cero.
- Pixel de fila, es el valor del pixel en la coordenada "y" de la imagen (no coordenada de longitud Y).

Un ejemplo de las lineas trazadas se muestra en la siguiente figura:

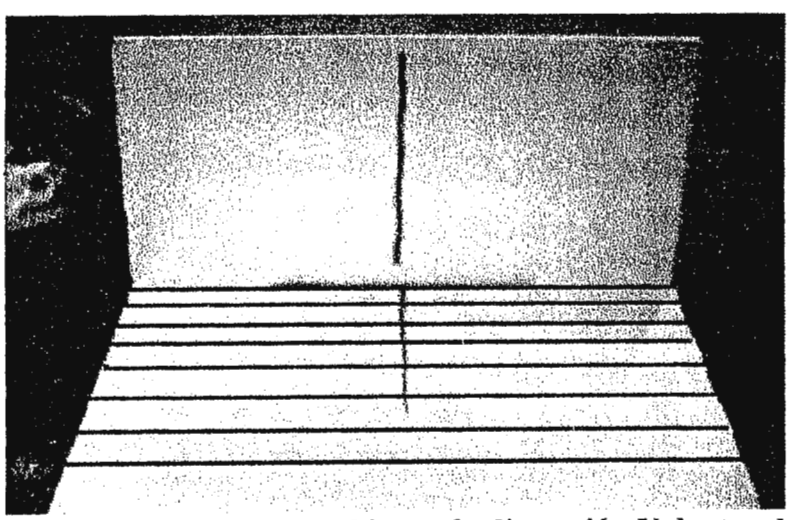


Fig. 46 Líneas que representan los cambios en la dimensión Y dentro del entorno.

Se coloca el contador de lineas en Y a cero. Se comienzan a leer los pixeles desde Yo (fila 103 en pixeles) hacia abajo, tomando como "x" constante la del centroide de la figura selecionada. Se buscan pixeles color negro pues estos representan las líneas; de encontrarse uno, se evalúa si es menor que la "y" del centroide, si la respuesta es verdadera se calcula el valor de la Y a partir de las siguientes fórmulas:

Py = (Ycentroide – coord. Linea anterior)/(coord.linea siguiente – coord. Linea anterior)

Ec. 20 Proporción menor a 1cm. tomado en coordenada Y.

La cual tiene como objetivo determinar una proporción que brinde los decimales correspondientes al caso en que se tenga el Y del centroide entre dos líneas (lo cual es el caso general). A este valor le llamaremos Proporción en Y (Py).
Finalmente, el valor de Y se obtiene de la siguiente suma:

Y = Contador de líneas en Y + Proporción en Y

Ec. 21 Valor final de la posición en Y.

4.5.3 COORDENADA Z.

Para el cálculo de la altura no es necesario el trazado de líneas, ya que se basa en el valor de longitud dado por Y. Esto se debe a que la altura sufre una distorsión dependiente de la distancia de la cámara con la figura u objeto captado. Este efecto indeseado se supera obteniendo una escala variable de la separación entre las líneas que representan a los centímetros. Esta escala es dependiente de la distancia antes mencionada. Lo anterior se ilustra en la siguiente figura:

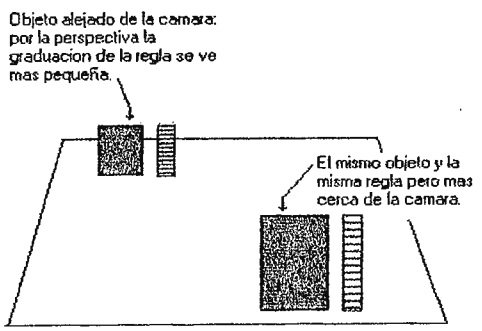


Fig. 47 Cambio en la altura con la perspectiva.

La escala antes mencionada se obtiene a partir de la siguiente ecuación obtenida expermentalmente y por medio de los mínimos cuadrados.

La ecuación que la rige es:

$$\textbf{Escala} = -5.127751 \times 10^{-5} Y^3 + 0.00390345\, Y^2 - 1.2885058 \times 10^{-4} Y + 10.9856$$

Ec. 22 Ecuación que permite compensar la escala de la coordenada Z.

Donde:

- Y es la longitud real en cm. que se toma desde el fondo hacia la cámara.
- Escala, es un factor multiplicativo.

La escala se aplica de tal manera que se resta el valor de la "y" del centroide (una posición de pixel) menos la coordenada "y" del pixel de la esquina inferior derecha. Esto nos da la cantidad de pixeles que representan la altura del objeto dentro de la imagen, pero no la altura (Z) real.

El valor de Z se calcula a partir de la expresión:

$$\textbf{Z} = -(\textbf{y centroide} - \textbf{y esquina inferior derecha}) * \textbf{escala}$$

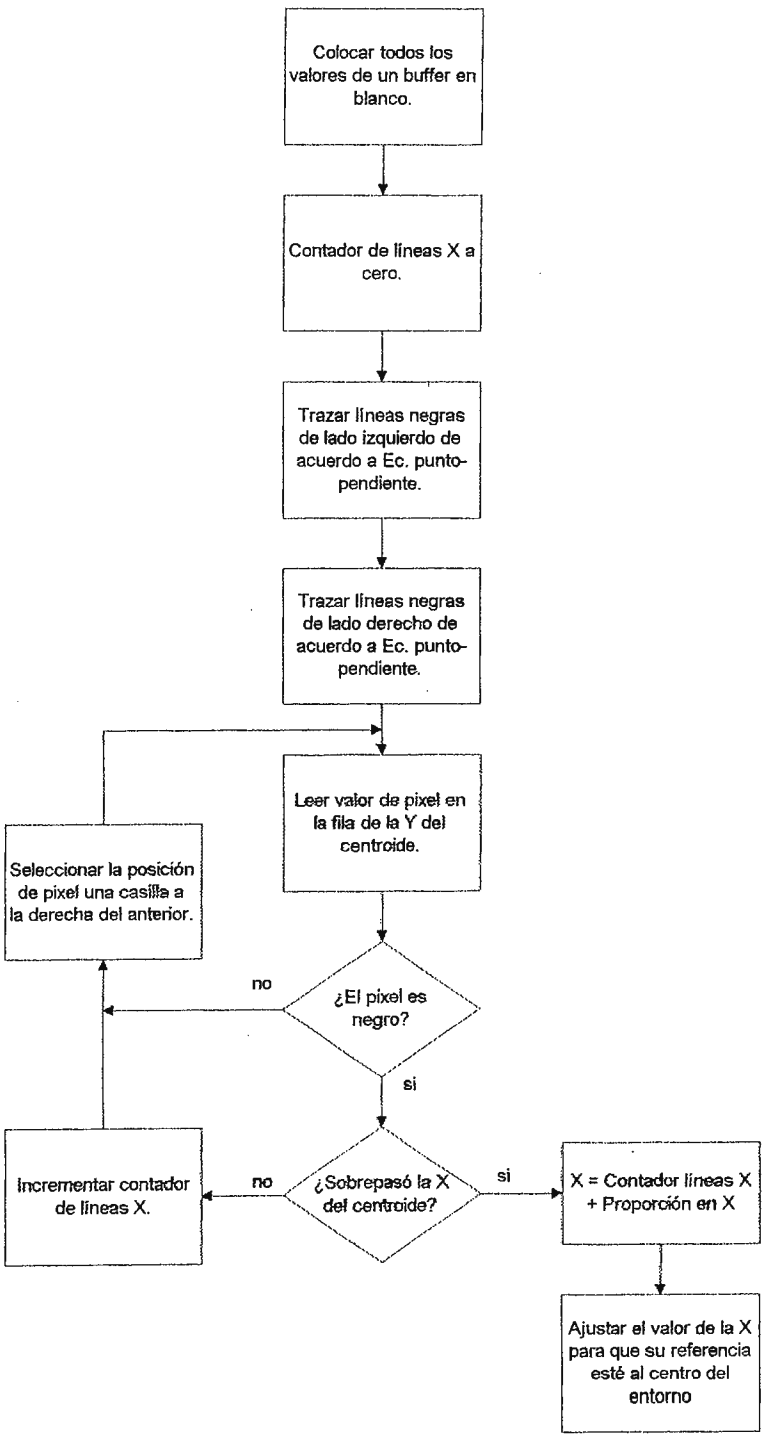
Ec. 23 Valor final de la posición en Z.

4.5.4 Cambio de coordenadas cartesianas a cilíndricas.

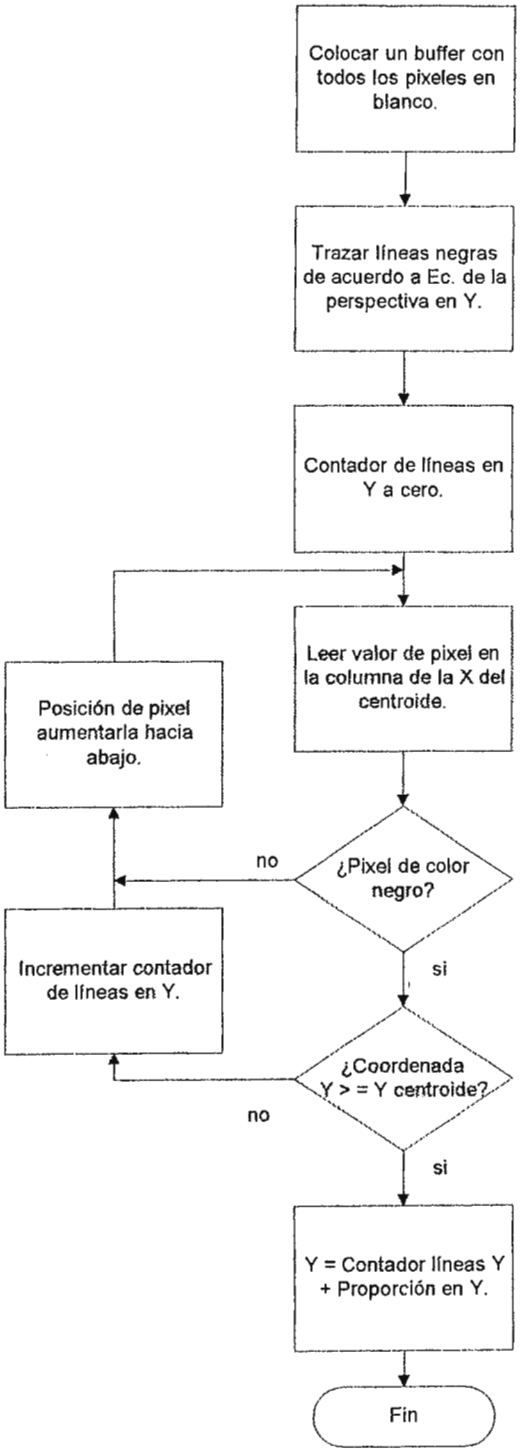
Los valores anteriores de X, Y y Z se cambian a cilíndricas debido a que los motores presentan una mayor facilidad de control usando este tipo de coordenadas. Esto se da porque los motores mueven (en forma individual) las piezas en posiciones angulares.

Los valores que se obtiene son un r que representa el valor del radio y un valor de ángulo θ que se usa para mover al motor de la plataforma giratoria del brazo (ambos obtenidos a partir de X y Y). El valor de Z se usa directamente.

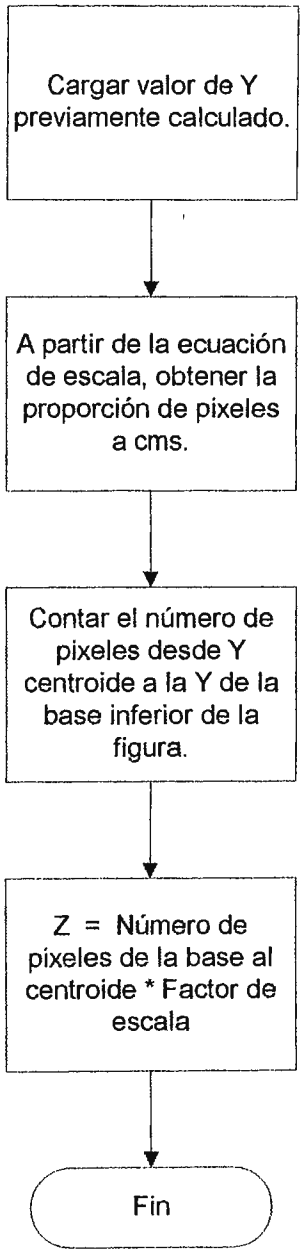
ALGORITMO PARA LA DETERMINACION DE LA POSICION X DEL OBJETO SELECCIONADO.



ALGORITMO PARA LA DETERMINACION DE LA POSICION Y DEL OBJETO SELECCIONADO.



ALGORITMO PARA LA DETERMINACION DE LA POSICION Z DEL OBJETO SELECCIONADO.



4. 6 DIAGRAMA DE FLUJO PARA EL MOVIMIENTO DE LOS MOTORES.

Para iniciar la explicación de los algoritmos de los motores es necesario conocer la ubicación y la nomenclatura usada para cada uno. La figura siguiente nos muestra al brazo desde dos proyecciones distintas en las cuales se identifican los motores desde CH#0 hasta CH#4.

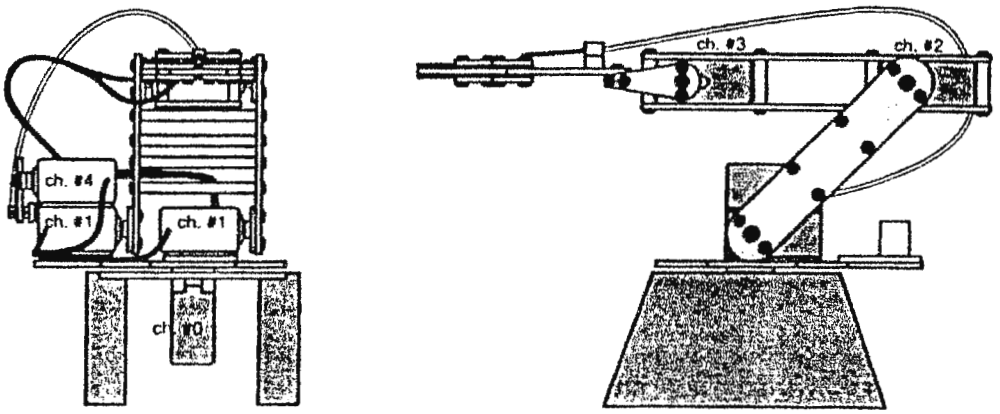


Fig. 48 Nomenclatura y ubicación de cada motor del brazo.

El movimiento de los motores se ha dividido en dos partes que se basan en el tipo de coordenada y la facilidad de control del brazo:

1. Movimiento angular (llamado en coordendas polares θ), es decir, el giro del tronco del brazo sobre el plano X,Y (Motor 0).

A partir de los valores de X y Y se calcula r y θ. El valor de θ se convierte a valores discretos entre 0 y 255 los cuales son necesarios para controlar al motor # 0. Este se encuentra en su posición central con el valor discreto de 130. Ver sección 3.4 para mayores detalles de las posiciones y del control del brazo.

La siguiente ecuación proporciona los valores discretos que se necesitan:

$$\mathbf{Pos\#0 = 130 - \theta / 0.418}$$

Ec. 24 Determinación de la posición del motor 0 (Base).

Donde:

- θ es el ángulo dado en grados, 130 es el valor con el cual el motor #0 coloca a la plataforma en la posición central (coordenada X=0).

- La constante 0.418 significa que el motor se mueve 0.418 grados cuando ocurre un cambio unitario en los valores discretos enviados a los motores.

2. Movimiento basado en ecuaciones polinomiales las cuales proporcionan la posición individual de cada motor (motores # 1, 2 y 3) para alcanzar el radio y la Z requeridos.

A partir del radio (19cm. <= r <= 38cm.) se determina la posición final de cada uno de los motores # 1, 2 y 3. Para esto se han deducido ecuaciones parametrizadas al radio utilizando ploteo de valores experimentales y el método de los mínimos cuadrados. Las ecuaciones dan valores correctos para determinados rangos, siendo controlados estos intervalos por medio de la función IF de Visual C++.

Las ecuaciones para cada motor son:

- Motor # 1.

$$\textbf{Pos \#1} = -0.17072177r^2 - 1.4039895r + 312.44215$$

Ec. 25 Ecuación para controlar el motor 1 (Antebrazo).

- Motor # 2.

Para 19cm <= r <=34cm.:

$$\textbf{Pos \#2} = -0.0038203r^4 + 0.371616r^3 - 13.828639r^2 + 228.327181r - 1141.784353$$

Ec. 26 Ecuación para controlar el motor 2 entre radios de 19 y 34 cms. (Brazo).

Para 34cm < r <= 38cm.:

$$\textbf{Pos \#2} = -1.8387978r^3 + 194.644703\ r^2 - 6874.5335r + 81112.80107$$

Ec. 27 Ecuación para controlar el motor 2 entre radios de 34 y 38 cms. (Brazo).

- Motor # 3.

Para 19cm <= r <= 30cm.:

$$\textbf{Pos \# 3} = -0.021992618r^3 + 1.2913284r^2 - 12.38749824r - 67.7121228$$

Ec. 28 Ecuación para controlar el motor 3 entre radios de 19 y 30 cms. (Muñeca).

Para 30cm < r <= 35cm.:

$$\textbf{Pos \# 3} = 0.41666667r^3 - 45.178571r^2 + 1612.619047r - 18847.857143$$

Ec. 29 Ecuación para controlar el motor 3 entre radios de 30 y 35 cms. (Muñeca).

Para 35cm < r <= 38cm.:

Pos # 3 = $-2.5136612r^3 + 266.948297r^2 - 9431.0637r + 110964.06998$

Ec. 30 Ecuación para controlar el motor 3 entre radios de 35 y 38 cms. (Muñeca).

Es importante mencionar que los valores obtenidos para cada posición a través de las ecuaciones deben de ser enteros (pues los motores se mueven con bytes de 00 a FF_{16}), por lo cual se hace necesario aplicar un truncado de la porción decimal de los valores calculados.

Dentro del algoritmo se manejan dos variables :

- "Actual" : la cual posee el valor discreto que representa la posición actual del motor.
- "Destino": posee el valor discreto fijado como meta y al cual el motor debe de llegar (posición final).

Es requerida una variable "actual" y "destino" para cada uno de los motores.

El primer paso que se realiza con los motores es llevarlos a una ubicación denominada "posición de inicio" cargándose esta posición en la variable actual de cada motor, los valores destinos deben de ser provistos por la imagen a través de los algoritmos antes mencionados.

Debido a que se requiere que los movimientos del brazo sean lo más naturales posible, se le envían datos a todos los motores de una manera casi simultánea; es decir, que en la práctica se observa que todos los motores se mueven al mismo tiempo. Además de las variables anteriores se utilizan 4 banderas (una por cada motor excepto el motor #4) las cuáles indican cuando cualquiera de los motores solicita moverse. Para iniciar el movimiento de los motores hacia su posición destino se evaluan las mencionadas banderas. Si alguna de ellas está en 1 se realiza la comparación entre el valor actual y destino. De no ser iguales indica que el brazo no ha llegado a la posición deseada por lo que debe de seguirse incrementando o decrementando de acuerdo a si el destino es mayor o menor que la posición actual del motor (la condición de la bandera de estado sigue en uno). Cada vez que se cambie de posición se debe de renovar la variable actual. Cuando la variable actual y la de destino son iguales se coloca la condición cero en la bandera de estado para indicar que el motor que ha llegado a su posición final. El proceso completo terminará cuando todos los motores lleguen a su posición final.

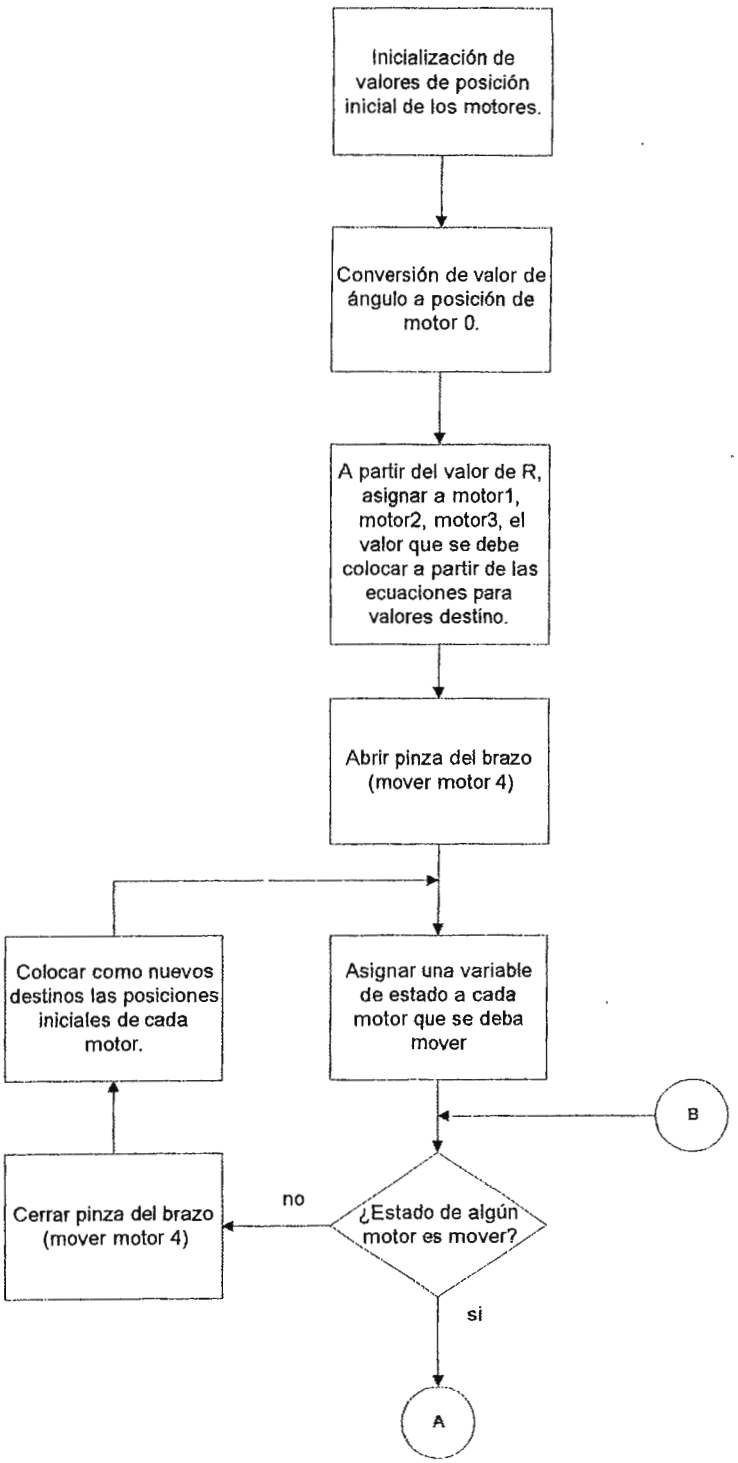
Cuando el motor llegue hasta el objeto seleccionado, el motor #4 se cerrará pues es el responsable de la posesión de los objetos por parte del brazo. Se realiza una evaluación justo cuando el brazo llega a su objetivo y si este tiene un ancho demasiado grande no será tomado

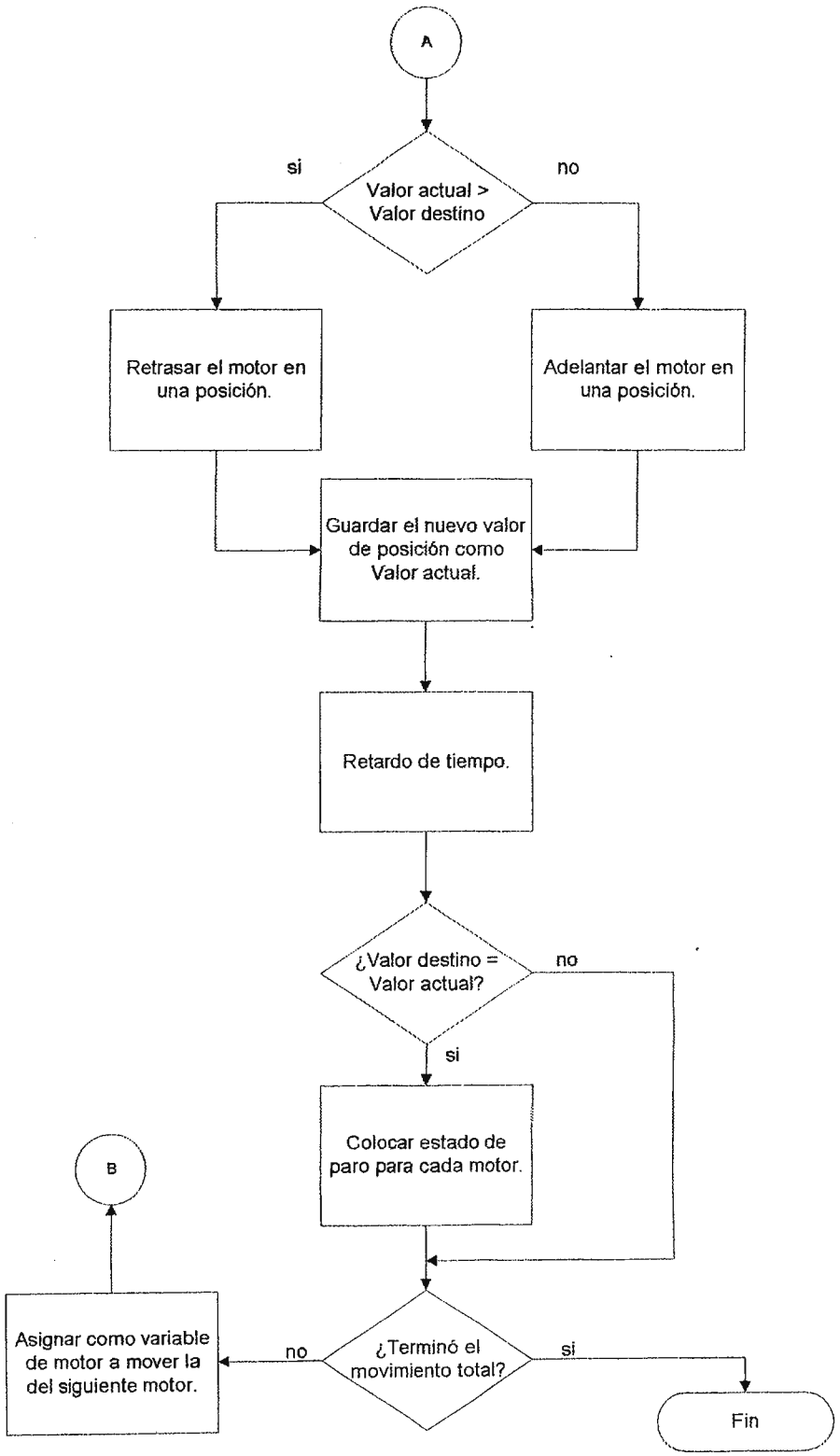
sino que el brazo simplemente se ubicará sobre él. Si tiene el objeto un ancho adecuado será tomado y el brazo lo levantará hasta llegar a sus posiciones de reposo.

Los objetos tienen como destino final una marca de color negro y dimensiones pequeñas que es también detectada por el sistema de reconocimiento de objetos. Debido a que esta marca puede estar en cualquier punto, se reasignan nuevas posiciones de destino para los motores del #0 al #3 y el agarradero del brazo se mantiene cerrado. Cuando el brazo llegue a esta marca suelta al objeto (se abre el agarradero) y lo coloca sobre esta señal.

Una vez realizada la colocación del objeto sobre la marca, el brazo recibe como nuevas coordenadas de destino, las mismas coordenadas de reposo que tenía antes de todos los movimientos.

ALGORITMO PARA EL CONTROL DE MOTORES.




A
si
Valor actual >
Valor destino
no
Retrasar el motor en
una posición.
Adelantar el motor en
una posición.
Guardar el nuevo valor
de posición como
Valor actual.
Retardo de tiempo.
¿Valor destino =
Valor actual?
no
si
Colocar estado de
paro para cada motor.
B
Asignar como variable
de motor a mover la
del siguiente motor.
no
¿Terminó el
movimiento total?
si
Fin

CAPITULO V.

SOLUCION GENERAL DEL PROYECTO.

5.1 GENERALIDADES.

Debido a que se toman imágenes de un mundo tridimensional pero se procesan sabiendo que solo tiene dos dimensiones; es necesario realizar una calibración de distancia y posicionamiento de la cámara con respecto al entorno, para que las cuadrículas virtuales correspondan a las dimensiones reales de la escena. De no cumplirse esto, los objetos serían reconocidos pero no se garantiza que el brazo se posicione adecuadamente sobre ellos.

Esta calibración se realiza de la siguiente forma:

1- Antes de correr el programa de calibración se normaliza la distancia entre el último borde del entorno y el lente de la cámara a 39cm., teniendo cuidado de que la cámara quede centrada a la escena.

2- Se toma una imagen del entorno sin el brazo robot presente y con tres líneas que forman una cruz. Tal como se muestra a continuación:

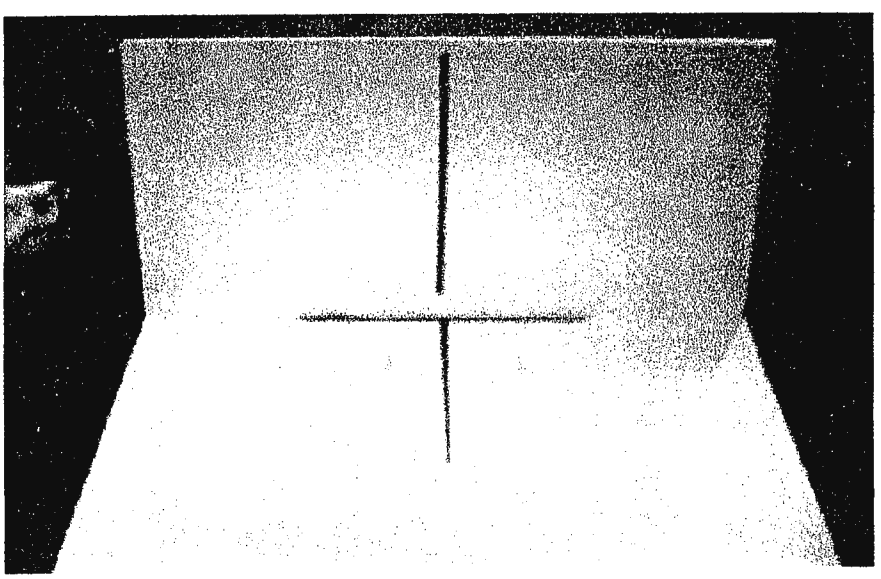


Fig. 49 Líneas de calibración del entorno.

3- El programa de calibración genera sobre la imagen una cruz en las posiciones en las que las líneas de referencia del entorno se deben encontrar.
4- Se procede a mover la cámara, sin alejarla del entorno sino paralelamente a él, hasta que converjan las líneas generadas por el programa con las líneas que están sobre el entorno.
5- Se experimenta con varias imágenes para tener la posición óptima de las lámparas y así obtener una iluminación adecuada al ambiente y la requerida por los algoritmos.

Luego de la calibración se procede a correr los programas de aplicación descritos a continuación.
Se desarrollaron tres aplicaciones (programas) diferentes para ejemplificar el desarrollo del reconocimiento de objetos. Lo anterior fue necesario ya que hay diferentes maneras de aplicar el reconocimiento de objetos y los algoritmos tienen variantes adecuadas a cada caso.

Estos programas son:

- ***Selección de objetos por área y por posición.*** Este programa determina de entre una multitud de objetos, uno a tomar en base al criterio de selección. El objeto es levantado y colocado en una marca de referencia que también es reconocida por el algoritmo. En este programa no se pueden detectar objetos traslapados.
- ***Seguimiento de objeto.*** Este programa realiza un reconocimiento cada cierto período, de tal forma que un objeto en movimiento es seguido por el brazo. La rapidez del movimiento del objeto no debe ser muy alta, pues el programa no es muy veloz debido a la cantidad de procedimientos necesarios para su ejecución.
- ***Determinación de objetos traslapados.*** Este programa es muy semejante al primero, solamente que tiene porciones de procesamiento de imágenes especiales, ya que debe reconocer objetos traslapados. Este programa presenta la desventaja de que los detalles de los objetos son resaltados hasta tal punto que estos son mal interpretados o algunos detalles se interpretan también como falsos objetos y el filtrado no se le aplica para no perder los ragos de los contornos.

5.2 SECUENCIA DE PASOS PARA EL PROGRAMA DE SELECCIÓN POR AREA Y POSICION DE LOS OBJETOS.

El programa comienza con una captura de una imagen del entorno, el brazo y los objetos. Después se procede a digitalizarla, colocándola en formato BMP dentro de un buffer previamente abierto y seleccionado. Una vez la imagen se tiene en memoria se procede a aplicarle un filtrado tipo mediana para eliminar el ruido. Aquí se dispone de una imagen aceptable para ser procesada.

Se requiere separar los objetos del fondo (entorno y brazo robot) por lo cual se le aplica umbralización. Los objetos se colocan en negro y el resto en blanco.

Teniéndo ya los objetos separados y claramente definidos se extraen las características de área relativa, centroide y coordenadas de esquina superior izquierda e inferior derecha del rectángulo que enmarca a cada uno de los objetos. Estos valores se guardan en un arreglo de datos.

Para seleccionar cual de todos los objetos va a ser tomado por el brazo se aplica una condición, de las dos siguientes:

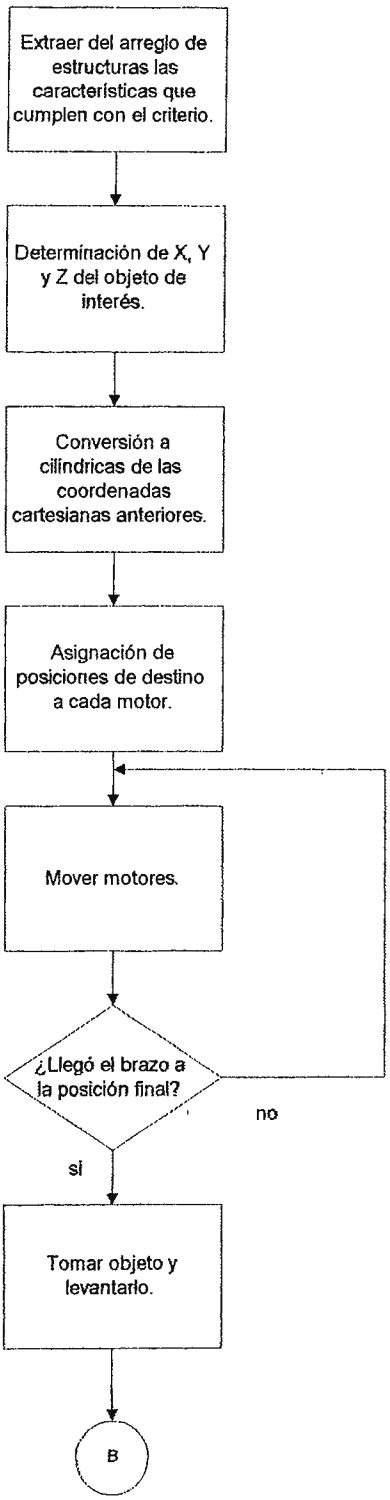
- Selección del que tenga mayor área.
- Selección del que tenga sus coordenadas de pixeles x, y en cierto intervalo dentro de la imagen (esto equivale a una selección por posición).

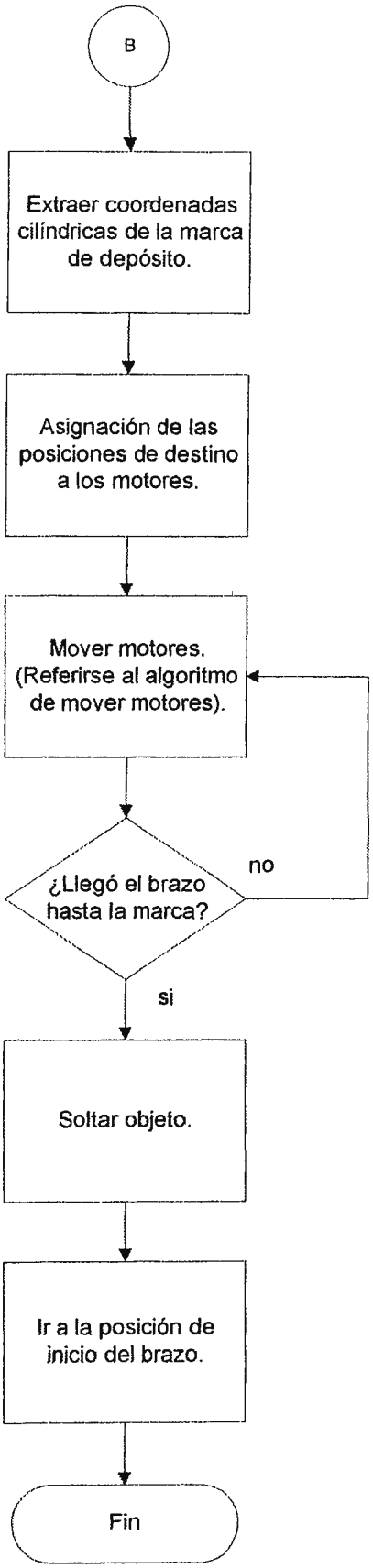
Las características de área y coordenadas (x, y) del centroide y las dos esquinas del rectángulo del objeto que cumplen la condición fijada son extraídas del arreglo que las contiene a todas. Estas características se usan para determinar la X, Y y Z de la posición real del objeto a partir de la imagen.

Teniendo estos valores se calculan las posiciones individuales en que cada motor debe moverse para llegar al objeto. Una vez alcanzado el objeto se le toma y levanta para colocarlo en la posición donde se encuentre una marca predeterminada.

Finalizada esta tarea el brazo regresa a las posiciones de inicio y está listo el programa para ser corrido de nuevo.

ALGORITMO GENERAL DEL PROGRAMA CON SELECCIÓN DE OBJETOS POR TAMAÑO O POR POSICION DENTRO DE LA IMAGEN.




B
Extraer coordenadas cilíndricas de la marca de depósito.
Asignación de las posiciones de destino a los motores.
Mover motores. (Referirse al algoritmo de mover motores).
¿Llegó el brazo hasta la marca?
no
si
Soltar objeto.
Ir a la posición de inicio del brazo.
Fin

5.3 SEGUIMIENTO DE LOS OBJETOS.

Este programa no puede tener más de un objeto en la escena y pretende ejemplificar el reconocimiento de objetos en movimiento.

Para este programa se siguen los siguientes pasos:

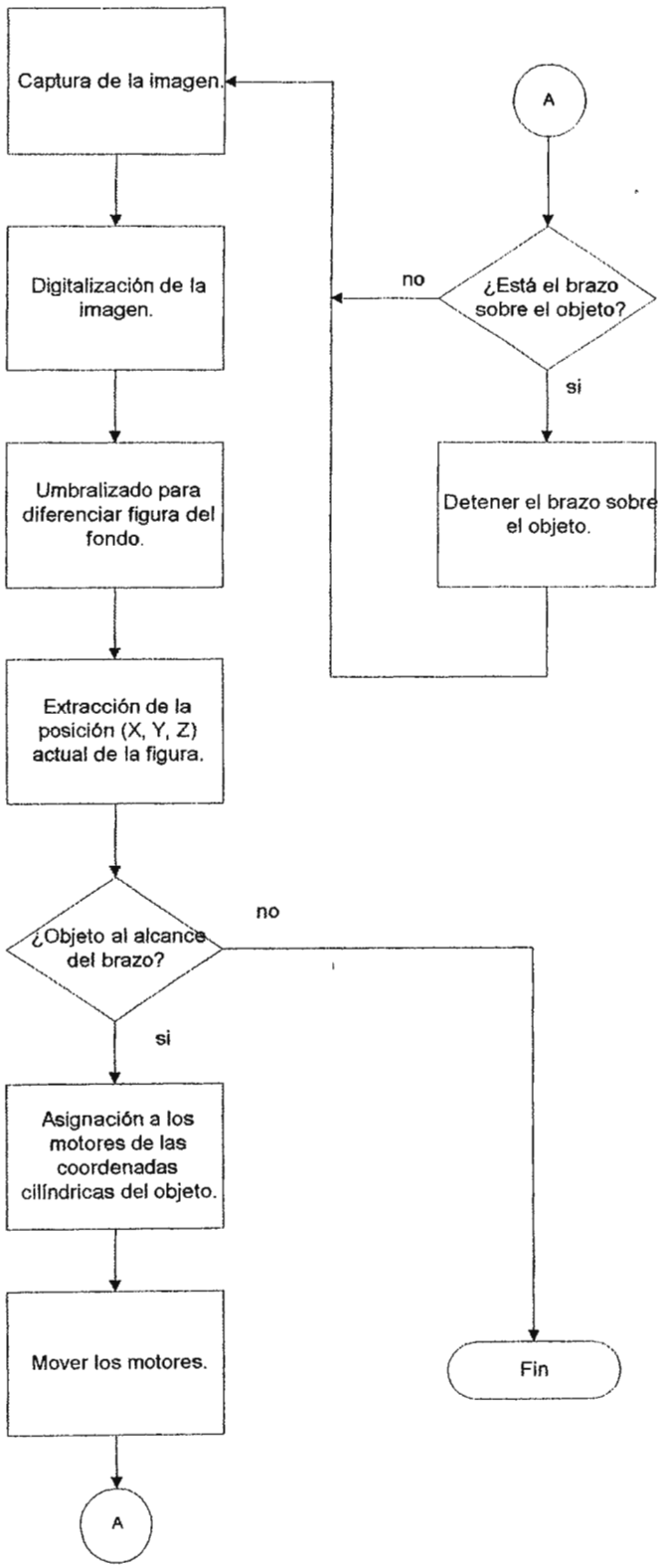
a) Captura y Digitalización de la Imagen.
b) Umbralización de la imagen para separar objeto del fondo.
c) Extracción de la posición X, Y y Z del objeto.
d) Se evalúa si las coordenadas anteriores se encuentran dentro del alcance del brazo. De encontrarse fuera de la región del alcance del brazo el programa finaliza, pues es imposible que lo siga.
e) Se envían los datos necesarios a los motores para que estos comiencen su movimiento. Es necesario especificar que el movimiento del brazo es por intervalos y que la rapidez de respuesta del programa a los cambios de posición del objeto es baja, pues el proceso de re-evaluación de la posición involucra una gran cantidad de pasos y además no es posible enviar datos demasiado rápido a los motores ya que estos se saturan.
f) El ciclo continua en el literal a) y terminará hasta que el objeto se coloque fuera del alcance del brazo robot.

Si el objeto ya no se cambia de posición el brazo sólo se posicionará sobre él, es decir, no lo tomará, aunque el programa siga corriendo. Si después de un reposo prolongado se mueve el objeto, se moverá el brazo para posicionarse sobre el objeto en la nueva posición.

Este algoritmo es la aplicación repetitiva del algoritmo que sirve para el programa de selección de objetos y por la gran cantidad de procesos que se realizan el programa tiene un límite de velocidad.

Este algoritmo debe su rapidez al hecho de que se manejan punteros y no arreglos para las imágenes, además el brazo se mueve una cierta cantidad de grados (10 posiciones consecutivas de cada motor) y después de este movimiento se vuelve a realizar una digitalización, repitiéndo todos los procesos hasta que la figura salga de los límites alcanzables del brazo.

ALGORITMO GENERAL DEL PROGRAMA DE SEGUIMIENTO DE OBJETOS.



5. 4 DETERMINACION DE OBJETOS TRASLAPADOS.

Este programa es muy semejante al de selección en cuanto a la secuencia del procesamiento que lleva, sin embargo, existe una diferencia fundamental: se aplican técnicas que resaltan los detalles para permitir que los contornos de objetos traslapados se puedan diferenciar.

El programa no aplica un filtro mediana, pues se reducen los detalles, sino que se comienza sacando dos copias de la imagen de la escena. Una de ellas se umbraliza, tal y como se hace con el programa de selección, y a la otra se le aplica la técnica Kirsch. Esta última técnica consiste en aplicar máscaras direccionales, cambiándose los valores de pixeles para aquellas en las que se dan transiciones con valores máximos. Recuérdese que un objeto en sus contornos presenta cambios bruscos de valor con respecto a los pixeles cercanos que forman parte del fondo. Para mayor detalle sobre el Kirsch ver la sección 2.3.4. Los contornos se colocan a negro y lo que es homogéneo se cambia a un valor de blanco.

Luego de tener ambas figuras se realiza una operación AND con las dos imágenes (la de umbralizado normal y la de Kirsch) y el resultado se guarda en otro buffer. La AND consiste en colocar en negro en la imagen resultante solamente aquellos pixeles que en ambas sean negro (a este se le asigna el valor lógico de 1) y aquellas regiones que en una o ambas imágenes tengan blanco se colocan a blanco (a este color se le asigna el valor lógico de 0).

Este algoritmo presenta una alta sensibilidad a los cambios en la iluminación y las sombras, de tal forma que mucha luz tiende a suprimir detalles y muy poca tiende a generar objetos falsos. Además, después de realizar lo antes descrito, es necesario hacer un proceso de erosión pues las figuras quedan unidas por pequeños "filamentos" debido a las sombras entre figuras. Al hacer esta erosión las figuras son distorsionadas, aunque se diferencien. Es por esto que se presentó como una aplicación aparte.

Una vez se tienen las figuras separadas (aunque algo diferentes de cómo son en realidad) se procede a extraer sus características de área, centro geométrico y coordenadas del rectángulo que enmarca a cada figura. Lo que sigue después de este programa es muy semejante al algortimo de la selección de objetos por medio de un criterio de selección.

La aplicación anterior no puede distinguir a un objeto muy pequeño que se encuentre detrás de un objeto mayor, esto es natural pues le sucede a un animal superior y aún al hombre mismo.

RECOMENDACIONES Y CONCLUSIONES.

RECOMENDACIONES.

I. Al utilizar en la cámara un lente que posea iris se tendrá la ventaja de regular la sensibilidad a la luz y así obtener independencia de las condiciones lumínicas. El sistema poseerá mayor versatilidad si se le automatiza por medio de un sensor de luz que controle la abertura del iris.

II. Si la aplicación necesita obtener detalles de ciertos sectores de la escena, se debe utilizar en la cámara un lente que posea *autozoom*. Este mecanismo del lente debe ser controlado por una PC, para lo cual se debe diseñar una interfaz.

III. Para proyectos en los que interese tener una visión 3D de los objetos es necesario usar visión *estereoscópica* y así poder tener una apreciación más exacta de la realidad. Esto permitirá tomar imágenes desde diferentes puntos y superponer la información.

IV. Es necesario escoger el tipo de brazo de acuerdo a las coordenadas a utilizar (como se muestra en la figura de la sección 3.4, figura 31).

V. Para hacer que un brazo robot tome las piezas de manera exacta (sin botarlos) se puede usar un detector de presión (que puede ser un simple interruptor) para saber si ya se ha tomado el objeto. Esto es agregar un "sentido del tacto" al sistema óptico de reconocimiento. También puede usar magnetos para sujetar piezas metálicas (que por lo general pesan más) sin que se deba realizar un aumento considerable en el torque.

VI. En aplicaciones que requieran movimientos rápidos del brazo es necesario tomar muy en cuenta la inercia del mismo.

VII. Para posicionar al brazo sobre los objetos sería mejor utilizar un algoritmo que proporcione la mejor ruta para acercarse tomando en cuenta la posición de la pieza con respecto a las demás.

VIII. Para hacer más flexible al sistema se puede diseñar una interfaz que determine el nivel de luminosidad, y con los valores que se obtengan de ésta se puede hacer que el programa haga un preacondicionamiento dinámico de la imagen. Se puede alterar el contraste y el brillo de la imagen para acondicionarlo a parámetros constantes y muy independientes de la luz ambiental.

IX. Como futuros proyectos, relacionados con la presente tesis, se sugieren:

- Un brazo robot de ensamble de piezas livianas que se deban colocar en posiciones diferentes sobre diversas formas de placas metálicas. Simulando una línea de ensamble de autos se puede usar una cámara para que el brazo "sepa" el tipo de pieza a usar según el auto a construir, que en este caso se identifica por la forma de la placa. El ensamble podría ser por soldadura o atornillado, según la factibilidad del brazo a usar.
- Un sistema de control de calidad para una imprenta o fotocopiadora que permita comparar la imagen producida con una patrón, y así determinar si se encuentra dentro de la tolerancia permitida.
- Una duplicadora de piezas que permita colocar como muestra un patrón para ser copiado. La máquina tomará diferentes vistas para extraer todas las dimensiones y fabricará una pieza idéntica a la muestra .
- Un sistema que permita evaluar de manera óptica la calidad de las pistas y los agujeros para componentes en los circuitos impresos.
- Un reconocedor de microorganismos que permita evaluar muestras de laboratorio (sangre, heces, orina, etc) para determinar la enfermedad del paciente basándose en la comparación de patrones de imágenes de bacterias conocidas.
- Un sistema de seguridad que se base en el reconocimiento óptico de huellas digitales o patrones únicos del ojo. Para reconocer y recordar patrones que puedan cambiar con el

tiempo (como es el caso del ojo) se necesita de un programa que evalue en base a probabilidades y pronóstico.

- La mejora a este proyecto es el desarrollo de un sistema que además de reconocer objetos, aprenda acerca de las figuras seleccionadas y las reconozca en cualquier posición que se encuentren en nuevas tomas de la escena. Es de agregar al sistema de este proyecto un control por redes neuronales que aprenda, recuerde y reconozca.

CONCLUSIONES.

I. Las coordenadas tridimensionales usadas para ubicar al brazo son obtenidas a partir de la posición de pixel de una imagen bidimensional, lo cual conduce a tener siempre un margen de error debido a que un pixel representa una infinita cantidad de puntos de la realidad.

II. Un pixel queda completamente definido por sus coordenadas (x, y) y por su valor de intensidad de color o gris.

III. El formato RS-170 es el más antiguo pero el más sencillo de utilizar. Es para niveles de gris, usando una resolución fija de 480 pixeles en la vertical (mientras que la horizontal es variable). Usa el método de *interlance*, es decir, que la imagen se realiza en dos trazos verticales traslapados. Cada trazo tiene 240 pixeles, dando una totalidad de 480 pixeles para el trazado completo de la imagen. Es mejor usar este formato por su simplicidad y aplicación común en las cámaras sencillas.

IV. En los sistemas de computadora se utilizan dos tipos de pixeles: de *imagen* y de *video*. Un pixel de imagen es el que se usa para representar en forma digital (dentro de memoria) a la escena; a estos pixeles se les realiza el procesamiento de imágenes. Un

pixel de video es la representación de la imagen para mostrarse en pantalla, por lo general varios pixeles de video representan un solo pixel de imagen y siempre residen en la porción de memoria de la controladora de video. Manipulando estos pixeles de video es como el sistema operativo controla diferentes tamaños de la representación en el monitor de la imagen.

V. Para eliminar ruido se usan, generalmente, filtros que tienden a uniformizar la imagen, lo cual lleva a una pérdida de las características de la imagen. Si lo que se desea es resaltar detalles, la mayoría de técnicas tenderán a resaltar también el ruido. Hay un compromiso entre tener poco ruido y tener mucha notoriedad de los detalles.

VI. La representación de una imagen digital es siempre una cuantización de los objetos y escenarios reales, por lo que tener una gran resolución permitirá acercarse más a la naturaleza continua de las cosas, es decir, mejor calidad de la imagen. Sin embargo, una imagen con resolución alta tendrá que utilizar más memoria y se requerirá un mayor tiempo de ejecución de los programas que la procesan.

VII. Dos imágenes pueden tener la misma resolución pero si una de ellas tiene color y la otra es de niveles de gris, la primera necesitará más memoria para su procesamiento que la segunda. Esto de debe a que la imagen a color usa 3 bytes por cada pixel (si fuera un formato RGB, por ejemplo) y la de niveles de gris (con valores de 0 a 255) necesita solamente un byte.

VIII. Los objetos más sencillos de manipular son aquellos que presentan características semejantes a una placa (planos y con grosor constante). Cuando se requiera usar objetos en los que se necesita conocer su profundidad se puede usar una de las siguientes alternativas: utilización de diferentes vistas (ya sea girando el objeto o teniendo varias cámaras), colocación de la cámara con un cierto ángulo de elevación con respecto a la

escena o el uso de luz dirigida desde diversas fuentes en diferentes ángulos para determinar profundidad a partir de su sombra.

IX. Seleccionar un fondo de color blanco presenta la desventaja de las sombras de los objetos (es desventaja si no se desean), pero permite que el brillo de la luz sea uniforme en todas las regiones, ya que de lo contrario estas deformaciones del fondo se pueden confundir con objetos.

X. Un fondo de color negro hecho de material metálico se comporta de manera muy diferente a uno del mismo color pero forrado de lona negra. La diferencia radica en que los materiales muy pulimentados tenderán a producir destellos que entorpecen el procesamiento de la imagen, mientras que los recubiertos con lona son rugosos y no generan destellos; siendo esto último lo más deseable. Por eso los objetos fueron forrados de lona negra en este proyecto.

XI. El desarrollo de los algoritmos que procesan las imágenes es mejor realizarlos por medio de funciones que se implementen dentro de archivos .H o DLL. Esto es recomendable ya que permite utilizar el mismo código muchas veces sin necesidad de redigitarlo, además de que el uso de DLLs permite cargar a memoria la función solamente cuando sea requerida. Esto libera una buena porción de memoria que se puede usar para otros propósitos.

XII. Para aplicar el procesamiento de imágenes es mejor no usar ningún algoritmo de compresión a la imagen, pues siempre se pierde cierta información y el procesamiento se vuelve más lento, por eso es recomendable usarlo como BMP (que es el formato más sencillo). La ventaja de aplicar compresión es la reducción del espacio requerido en memoria y en disco.

XIII. Para determinar expresiones matemáticas empíricas que describan un fenómeno de manera exacta (como el valor de posicionamiento de cada motor en función del radio de alcance) es conveniente realizar un ploteo de valores fijos obtenidos experimentalmente y a partir de ellos hacer una aproximación sucesiva por iteración o por medio del método de mínimos cuadrados a un polinomio que incluya los valores del experimento muy aproximadamente. En muchas ocasiones estas ecuaciones corresponden mejor a la realidad si se realizan por secciones.

XIV. Para determinar la posición de las imágenes dentro de la escena es necesario usar un cierto tipo de referencia dentro de la imagen misma, que permita evaluar las distancias tomando en cuenta las distorsiones debidas a la perspectiva. Estas referencias pueden ser cuadrículas imaginarias que sigan la tendencia de las deformaciones propias de la escena.

XV. Al realizar un programa que realice un reconocimiento de objetos en movimiento se debe de tomar en cuenta que es necesario hacer digitalizaciones continuas y aplicarles los algoritmos de manera repetitiva, por lo cual existe un mínimo tiempo de respuesta que determina la velocidad máxima a la que se moverán los objetos.

XVI. Para realizar el reconocimiento de objetos traslapados es necesario enfatizar todas las transiciones de pixeles (los cambios de figuras) en la imagen. Luego estos bordes resaltados se usan para "cortar" las figuras traslapadas que previamente han sido separadas del fondo (segmentadas). Estos algoritmos tienden a distorsionar la imagen debido al proceso de separación que se realiza. A semejanza del ojo de los seres vivos no se puede distinguir un objeto cuando otro lo tape o lo traslape de forma tal que pierda sus características sobresalientes.

BIBLIOGRAFIA.

1. **THE IMAGE PROCESSING, HANDBOOK.** Russ, John.
   Segunda Edición; IEEE Press.

2. **MANUAL DEL VIDEO AFICIONADO.** Cheshire, David.
   Segunda Edición, Libros CUPULA.

3. **LA CAMARA CREATIVA DE VIDEO.** Vidal, Albert.
   Primera Edición, Libros CUPULA.

4. **DIGITAL IMAGE PROCESSING ALGORITHMS.** Pitas, I.
   Año 1993, PRENTICE HALL.

5. **SISTEMAS DE COMUNICACIONES ELECTRONICAS.** Tomasi, W.
   Año 1996, PRENTICE HALL.

6. **www.ask.com** Motor de búsqueda de otros buscadores. Permite buscar temas exactos de investigación o preguntas completas, dando como respuesta una lista de sitios que se encuentran en casi todos los demás buscadores.

7. **spib. rice.edu / spib.html** Información variada de procesamiento de imágenes.

8. **www.optics.arizona.edu** Departamento de la Universidad de Arizona sobre imágenes.

9. **www.ph.kcl.ac.uk** Grupo de Procesamiento de Imágenes de King's College London. Presenta conferencias en formato PS.

10. **www.codeguru.com** Centro de consulta a desarrolladores y expertos en programación en Visual C, Visual Basic, Unix, Java, etc.

11. **www.vsipl.org/** Recursos sobre programación para procesamiento de imágenes.

12. **sipi.usc.edu** Instituto de Procesamiento de Imágenes y Señal de la University Southern California. Tiene archivos diversos sobre procesamiento de imágenes.

13. **eleceng.ukc.ac.uk/~rls3/Frame1.htm** Información sobre cámaras, lentes, formatos de video y color. Documentos y revistas Online publicadas por University of Kent.

14. **nexus.uta.edu/eeweb/ip/index.html** Cursos impartidos por la Universidad de Texas (en Arlington) sobre Procesamiento de Imágenes y RedesNeuronales (en formato PDF).

15. **noodle.med.yale.edu/intro.html** Departamento de Diagnóstico, Radiología e Ingeniería de la Escuela Yale de Medicina. Presenta comentarios y ejemplos visuales de imágenes médicas procesadas.

16. **www.cn.purduc.edu/NSIP/** Conferencias sobre Procesamiento de Imágenes y Señales no Lineales (en formato Post-Script).

17. **www.lynxmotion.com** Sitio Web comercial de venta de robots para usos científicos o didácticos.

18. **www.epixinc.com** Sitio Web comercial de venta de diferentes tipos de tarjetas digitalizadoras de video.

19. **www.eecs.wsu.edu/Ipdb/title.html** Base de datos sobre ejemplos de algoritmos de procesamiento de imágenes. Universidad de Washington.

ANEXO A.
CARACTERISTICAS DE LA CAMARA.

MICROLENS SENSOR TECHNOLOGY!

HIGH PERFORMANCE MONOCHROME CCD CAMERA

4910 SERIES

High Resolution
1/2" On-Chip-Microlens Interline Transfer

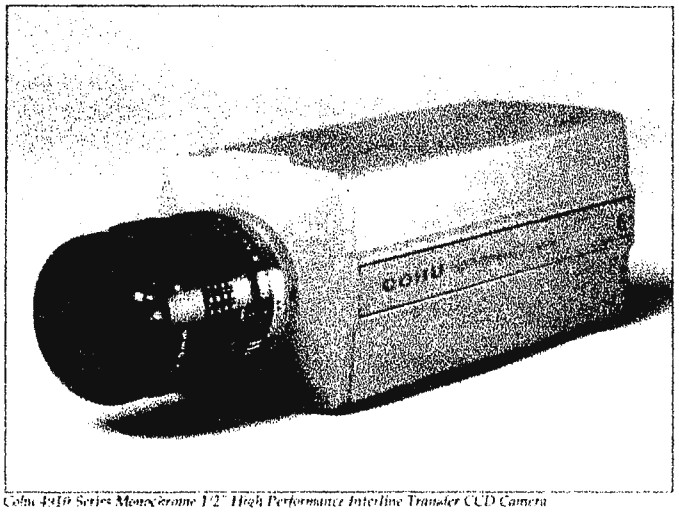
The 4910 Series High Performance Monochrome 1/2" CCD Cameras from Cohu offer high resolution and high sensitivity for use in a broad range of security/surveillance, scientific, and industrial video applications.

The 4910 Series cameras feature a 1/2"-format on-chip microlens sensor, which reduces dark current, lag, and blooming, while improving dynamic range and spectral characteristics. For video applications prone to streaking problems, a 1000:1 overload capability allows transmission of clear video signals even when bright incidental light is present in the scene.

The 4910 Series design also incorporates a removable trim plate for side panel access to controls such as gamma, electronic shutter, and gain.

Available in RS-170 and CCIR models, the 4910 Series cameras feature 26 dB of AGC for high sensitivity in low light-level applications. They are rugged, yet lightweight and compact, making them ideal for easy system integration. And 4910 Series cameras are backed by a full two-year warranty.

A leading U.S. manufacturer of closed circuit video cameras and systems for more than 40 years, we welcome requests for special products and complete CCTV systems.



Cohu 4910 Series Monochrome 1/2" High Performance Interline Transfer CCD Camera

APPLICATIONS

- **Security/Surveillance**
  - Military Installations
  - Nuclear Power Plants
  - Hazardous Waste Management
  - Traffic Management
  - Airports
  - Mass Transit Systems
  - Radar Tracking Systems
- **Image Processing**
- **Machine Vision**
- **Process Control**
- **Quality Control**
- **Image Analysis**

FEATURES AND BENEFITS

- **High Resolution** — for better definition, error-free results
- **Side-Panel Controls** provide convenience and precision
- **1/2" On-chip-microlens Interline Transfer Imager** virtually eliminates overload streaking, improves dynamic range
- **Eight-Speed Electronic Shutter** reduces blurred images of fast-moving objects
- **High Sensitivity** permits operation over a broad range of light levels
- **Choice of Synchronization Options** for greater versatility
- **High Signal-to-Noise Ratio** for clear, noise-free video
- **Asynchronous Reset** provides random vertical reset capability for production line applications
- **Optional Electronic Iris** automatically controls exposure from 1/60 sec. to 1/15,000 sec.
- **Blemish-Free Imager** — no dead pixels
- **Made in U.S.A.** — direct factory support
- **1000:1 Overload Capability** permits incidental light overloads up to ten times that of other CCD cameras
- **No Lag or Image Retention** — provides fast, clean, precise images
- **Zero Geometric Distortion** for consistent corner-to-corner linearity
- **26 dB AGC** for increased sensitivity at low light levels
- **Optional IR Filter**
- **"C" or "CS" Lens Mount** expands your choice of lenses
- **Top or Bottom Mounting** for easy installation
- **State-of-the-Art Design and Construction** for total, solid-state reliability and long life
- **Choice of RS-170 or CCIR Models**
- **Two-Year Warranty**

Designed and Manufactured in U.S.A.

COHU
Cohu, Inc./Electronics Division

4910 SERIES HIGH PERFORMANCE MONOCHROME CCD CAMERA

ORDERING INFORMATION

491X— X X X X / XXXX

Power Options

2 12V ac or dc
3 230V ac, 50 Hz, with ac wall adapter (CCIR Models)
4 24V ac or dc
5 115V ac, 60 Hz, with ac wall adapter (RS-170 Models)

Sync Options

2 Genlock* (revert to crystal) RS-170
3 Genlock*(revert to variable phase line lock) RS-170
4 Asynchronous Reset RS-170
5 Genlock* (revert to crystal) CCIR
6 Genlock* (revert to variable phase line lock) CCIR
7 Asynchronous Reset CCIR

* Genlock can be composite sync or separate H & V Drive

Optical Filters

0 None
1 IR Filter (Non-removable)

Options

0 None (Standard TV Rate)
1 Frame Mode
3 Electronic Iris*

* Electronic Iris option is designed for use with manual iris lenses only. With this option, the camera operates in the field integration mode. Use of the electronic iris defeats electronic shutter positions.

Module Options

0 None

Lens Options

Manual Iris, CS Mount
A003 3.7mm, f/1.6, 1/2"
A006 6mm, f/1.4, 1/2"
A013 12mm, f/1.4, 1/2"

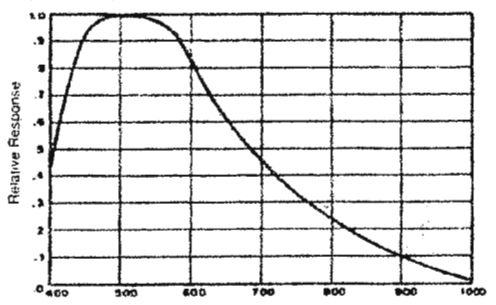
Manual Iris, C Mount
***AL04** 4.5mm, f/2.0, 2/3"
***AL08** 8mm, f/1.4, 2/3"
AL16 16mm, f/1.4, 2/3"
AL25 25mm, f/1.4, 1"
AL50 50mm, f/1.4, 1"
AL75 75mm, f/1.8, 1"
* Wide Angle

Auto Iris, CS Mount
EH04 3.7mm, f/1.6, 1/2"
EH06 6mm, f/1.4, 1/2"
EH13 12mm, f/1.4, 1/2"

Auto Iris, C Mount
ES04 4.2mm, f/1.8, 1/2"
ES05 4.8mm, f/1.8, 2/3"
ES08 8mm, f/1.4, 2/3"
ES12 12.5mm, f/1.4, 2/3"
ES16 16mm, f/1.4, 2/3"
ES25 25mm, f/1.4, 1"
EH35 35mm, f/1.4, 2/3"
ES50 50mm, f/1.4, 1"
EH75 75mm, f/1.8, 1"
Please consult factory for other lens selections.

COHU RESERVES THE RIGHT TO CHANGE SPECIFICATIONS WITHOUT NOTICE.

SPECTRAL RESPONSE



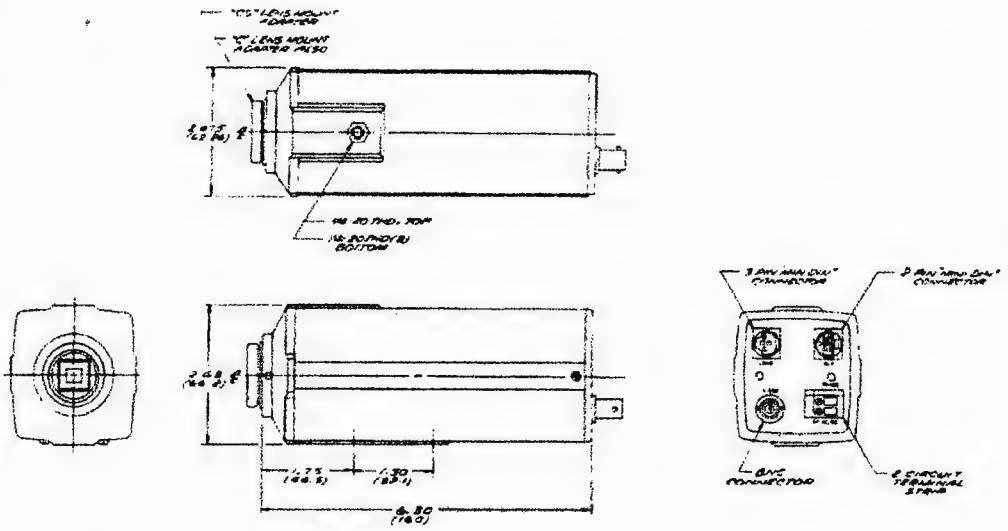
ADVANCED VIDEO TECHNOLOGY
Made in the U.S.A.

5755 Kearny Villa Road • San Diego, CA 92123
Telephone: (619) 277-6700 • FAX: (619) 277-0221

COHU
Cohu, Inc./Electronics Division

4910 Series - Printed in U.S.A. - 5/98

4910 SERIES DIMENSIONS



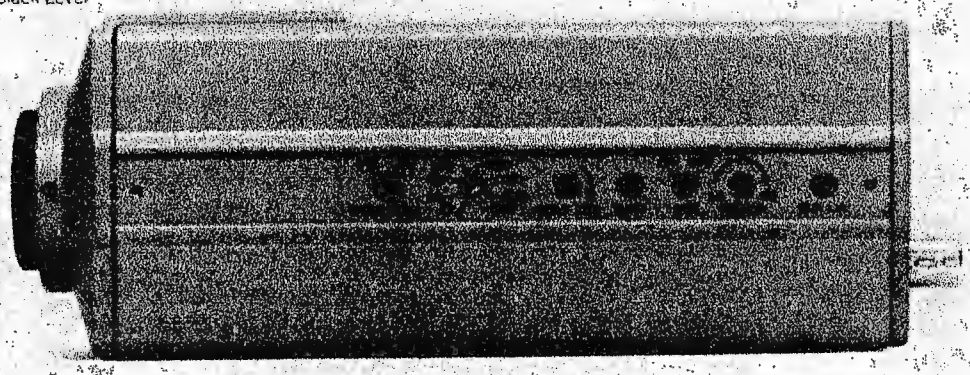
NOTE: DIMENSIONS IN INCHES (MM)

Figure 1

SIDE PANEL CONTROLS

An easily removable trim plate allows access to the following side-panel controls:

- Electronic Iris ON/OFF
- Eight-Step Shutter Timing
- AGC Peak/Average
- Gain
- AGC ON/OFF
- Gamma
- Black Level



4910 HIGH PERFORMANCE MONOCHROME CCD CAMERA

ELECTRICAL

Image Area
6.4 x 4.8 mm (corresponding to 1/2" image tube)
Active Picture Elements
RS-170: 768H x 494V
CCIR: 752H x 582V
Imager Type
On-chip microlens sensor interline transfer CCD
Cell Size
RS170: 8.4 x 9.8 microns
CCIR: 8.6 x 8.3 microns
Resolution
RS170: 580 horizontal TVL, ≥350 vertical TVL
CCIR: 560 horizontal TVL 450 vertical TVL
Sensitivity (faceplate) @2850 K
Please see Table 1
Electronic Shutter
Eight steps from 1/50 or 1/60 to 1/10,000 second (1/50 or 1/60, 1/125, 1/250, 1/500, 1/1,000, 1/2,000, 1/4,000, 1/10,000 second)
Integration
Integration period controllable through external input pulse
Grab pulse output
Field (1/60 or 1/50 second) or Frame (1/30 or 1/25 second) integration selected by internal jumper
Video Output
1.0 V p-p @75 ohms, unbalanced
Gamma
variable 0.45 to 1.0
AGC
26 dB, variable gain
Signal-to-Noise Ratio
≥56 dB at gamma 1, gain 0 dB 38 dB at gamma 1, AGC On
Auto Lens
Separate lens video ratio tracks AGC peak/average adjustment to eliminate AGC/auto lens interaction
Power: +15V, 35 mA maximum
Synchronization
Genlock, revert to variable phase line lock with zero crossing detector
Genlock, revert to crystal
Crystal Lock
H & V Drive
Asynchronous Reset
Internal Clock Speeds
RS170: 28.6363 MHz
CCIR: 28.375 MHz
Power Requirements
12V ac or dc (standard)
24V ac or dc (optional)
115V ac (optional on RS-170 models, includes wall transformer and connector)
230V ac (optional on CCIR models, includes wall transformer and connector)
4.2 watts dc power consumption
LED Power Indicator, Green

MECHANICAL

Dimensions (less lens)
Please see Figure 1
Weight (less lens)
18.5 ounces (0.52 kg)
Lens Mount
'CS' mount 16mm format
'C' mount with adapter (furnished)
Camera Mounts
1/4 - 20 threaded holes top and bottom
Connectors
Video (BNC)
Power (2 circuit screw terminal)
Lens (3 pin Mini-DIN)
External Sync (8 pin DIN)
Pin 1 External Vertical Trigger In
Pin 2 External Sync/Horizontal Trigger In
Pin 3 Grab Pulse Out (-)
Pin 4 Ground
Pin 5 Ground
Pin 6 Vertical Reset In
Pin 7 Grab Pulse Out (+)
Pin 8 Integrate Input

ENVIRONMENTAL

Ambient Temperature Limits
Operating -20 to 60° C (-4 to 140°F)
Storage -30 to 70°C (-22 to 187°F)

Humidity
Up to 95% relative humidity

Vibration
Sine vibration from 10 to 2,000 Hz, 5G peak, all 3-axis, 1/2 hour per axis per MIL-E-54007, para 3.2.24.5.1.2, fig. 2, curve IIIA
Random vibration from 10-2,000 Hz, 11G RMS all 3-axis, 1/2 hour per axis, meets MIL-E-5400T, para 3.2.24.5.1.2A, category 6

Shock
Up to 15 G-in any axis under nonoperating conditions.

SENSITIVITY

	Full Spectrum	With IR Filter
Full Video, No AGC	0.065 fc (0.65 lux)	0.25 fc (2.5 lux)
80% Video, AGC On	0.002 fc (0.02 lux)	0.01 fc (0.1 lux)
30% Video, AGC On	0.0004 fc (0.004 lux)	0.0015 fc (0.015 lux)

Table 1

This model has been tested and found to comply within the FCC limits for Class 'B.'

ANEXO B. ESPECIFICACIONES DE TARJETA DIGITALIZADORA.

PIXCI- SV4

INNOVATIONS

Color and Monochrome Video
Formats Supported:
S-Video, NTSC, RS-170, CCIR, PAL PCI Bus Master Real-Time Transfer to PCI Bus Crop, Scale, and View in a Window Automatic Format Detection 4 Input Triggers and 4 Output Triggers Programmable Hue, Brightness, Saturation, and Contrast Plug 'N Play Operation Extensive Software

APPLICATIONS

- Automated Inspection Motion.
- Analysis Microscopy Medical.
- Imaging Robotics.
- Laser.
- Beam Analysis.
- Object Tracking Multimedia.
- Print Quality Inspection.

The **PIXCI® SV4** imaging board, for the PCI bus, is designed to take advantage of the power of the host computer.
Applications which were once restricted by limited memory or processing power can now be easily accomplished with the **PIXCI® SV4** board and a compatible PCI computer.

ACQUISITION

A unique digital genlock circuit ensures precise synchronization of every image. The **PIXCI® SV4** board automatically recognizes unstable signals (for example, from a VCR) and adapts its locking mechanism to accommodate the source.
A multiplexer allows software selection of either a composite or an S-Video source. Programmable automatic gain, hue, brightness, saturation, and contrast adjustments condition the video signal.
Image sequences may be captured at full or reduced frame rates, onto the PCI bus, for storage in the host computer's memory, or passed to other devices on the PCI bus such as disk controllers or VGA adapters.

SCALING AND CROPPING

The window of video to be captured may be cropped in single pixel increments, then scaled in ratios from 1:1 to 1:14, down to as few as 4 pixels by 1 line of image data. Horizontal and vertical scaling is performed in real-time by interpolation, providing an accurate representation of the original image.

DISPLAY

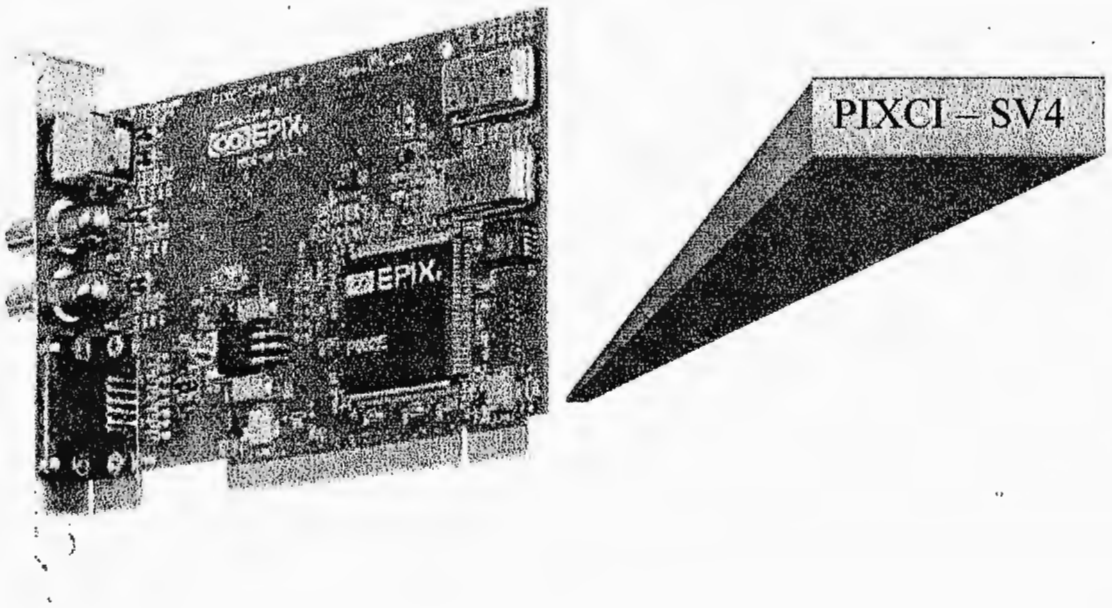
Depending on the VGA adapter, 24 bit RGB color images or 8 bit monochrome images may be displayed. The full, scaled, or cropped image may be placed anywhere on the VGA screen. Luminance or monochrome image data can be passed directly to the VGA for live video-in-a-window display. Color images are normally stored in the host computer's memory, converted into RGB data, then displayed on the VGA adapter. With a fast processor, fast PCI bus, and fast VGA adapter, live color image data may be displayed. If the VGA adapter can accept Y/C color pixels, then images can be sent directly to the VGA adapter across the PCI bus.

TRANSFER RATES

The PIXCI® SV4 board is designed to use the 132 Mbytes/sec burst mode transfer rate of the PCI bus. As a bus master, the PIXCI® SV4 board sends image data to the PCI bus; it does not wait for the computer's CPU to read images from the board into PC memory.

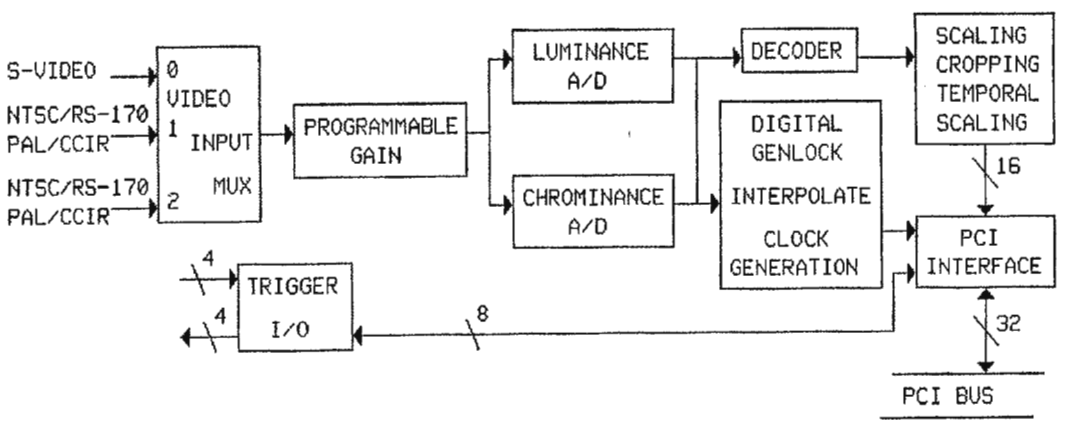
I/O CONTROL

Four input and four output TTL trigger signals are available for synchronization with external events.



FEATURES	
MUX - The multiplexer selects the video source for the Programmable Gain from either the S-Video input connector or from the composite video source on the BNC connector. The multiplexer may be switched during vertical blanking. **Programmable Gain** – Compensates for reduced amplitude in the analog signal input. Gain can be programmed from 0% to more than 200%. **Luminance A/D** – Provides analog to digital conversion of NTSC, RS-170, CCIR, PAL, and the luminance (Y) component of S-Video sources. **Chrominance (Color) A/D** – Provides analog to digital conversion of the color (C) component of S-Video.	**Decoder** - Separates the Y/C components. Generates the U/V color difference signals. **Digital Genlock** - Automatic synchronization circuitry for precise digitization. Accommodates video sources which have variable periods, such as video tape recorders. Generates the pixel clock for transferring image data to the PCI bus interface. **Scaling, Cropping** - Interpolation is used to scale images to 1/14 of their original size. **Temporal Scaling** - Image sequences may be captured at full or reduced frame rates. **Trigger I/O** – Four input and four output TTL triggers can be used for synchronization with external events. The trigger signals are controlled by the host CPU.

DIAGRAM BLOCK.



HARDWARE

SPECIFICATIONS

CONNECTIONS:

- 4 Pin DIN: S-Video Input
- Two BNC-Jacks: Composite Video Inputs
- DB15: TTL I/O Triggers
- Cables optional

VIDEO INPUT:

- Color or Monochrome Video Acquisition: S-Video, RS-170, CCIR, NTSC, PAL
- Resolution-Pixels: 754x480: RS-170, NTSC, S-Video 922x580: CCIR, PAL, S-Video
- Resolution-Depth: 8 bit: RS-170, CCIR YUV [4:2:2]: NTSC, PAL YCrCb: S-Video
- Capture/Display Rate: 30 fps: RS-170, NTSC, S-Video 25 fps: CCIR, PAL, S-Video

DATA FORMATS TO PCI BUS:

- Monochrome 8 bit
- YCrCb (UYUV, YUV4:2:2): 16 bit
- RGB: 24 or 32 bit
- Capture Rate:
  30 fps: RS-170, NTSC, S-VIDEO
  25 fps: CCIR, PAL, S-VIDEO

TRANSFER RATES:

- Requires a PCI motherboard with burst mode to host memory data rates of at least 30 MB/S.
- Contact EPIX or an EPIX distributor for suggested motherboards.

DISPLAY – Windows:

- Display resolution as per installed VGA device driver.
- A DCI compatible S/VGA adapter is required for real-time display.

DISPLAY - DOS:

- Via Standard VGA: limited to 4 bits (16 gray levels), non real-time display.
- Via Super VGA: 8 bit, 256 gray level display. Color display via adapters supporting 24 bit RGB. S/VGA adapters must be VESA 1.0 compatible.
- Contact EPIX or an EPIX distributor for suggested S/VGA adapters.

BUS REQUIREMENTS:

- 32 bit PCI Bus Master slot
- 0.55 Amps @ +5 Volts
- 4.913 in. by 3.350 in.

PRICE LIST

Effective Date: 15 February 1998

Hardware

ORDER CODE	DESCRIPTION	UNIT PRICE
PIXCI® SV4	PIXCI® SV4 Imaging Board, PCI Bus, has one female S-Video connector and two BNC jacks for multiplexed video input.	$ 395
SVID-4DINP-6FT	Cable, S-Video, 4 pin plug at each end, 6 ft.	$ 25
BNCP-BNCP-6FT	Cable, BNC plug to BNC plug, 6 ft.	$ 25
DB15-2-BNCP-6FT	Cable, DB15 to 2 BNC plugs, XD0 (ext. in), XD4 (ext. out), 6 ft.	$ 50
PC-xxxxxx	PC w. Intel Pentium II & motherboard, AGP. EPIX Qualified for PIXCI® SV	Call for prices

www.epixinc.com

ANEXO C.
CARACTERISTICAS DE LA UART 16550D

8.0 Registers

The system programmer may access any of the UART registers summarized in Table II via the CPU. These registers control UART operations including transmission and reception of data. Each register bit in Table II has its name and reset state shown.

8.1 LINE CONTROL REGISTER

The system programmer specifies the format of the asynchronous data communications exchange and set the Divisor Latch Access bit via the Line Control Register (LCR). The programmer can also read the contents of the Line Control Register. The read capability simplifies system programming and eliminates the need for separate storage in system memory of the line characteristics. Table II shows the contents of the LCR. Details on each bit follow:

Bits 0 and 1: These two bits specify the number of bits in each transmitted or received serial character. The encoding of bits 0 and 1 is as follows:

Bit 1	Bit 0	Character Length
0	0	5 Bits
0	1	6 Bits
1	0	7 Bits
1	1	8 Bits

Bit 2: This bit specifies the number of Stop bits transmitted and received in each serial character. If bit 2 is a logic 0, one Stop bit is generated in the transmitted data. If bit 2 is a logic 1 when a 5-bit word length is selected via bits 0 and 1, one and a half Stop bits are generated. If bit 2 is a logic 1 when either a 6-, 7-, or 8-bit word length is selected, two Stop bits are generated. The Receiver checks the first Stop-bit only, regardless of the number of Stop bits selected.

Bit 3: This bit is the Parity Enable bit. When bit 3 is a logic 1, a Parity bit is generated (transmit data) or checked (receive data) between the last data word bit and Stop bit of the serial data. (The Parity bit is used to produce an even or odd number of 1s when the data word bits and the Parity bit are summed.)

Bit 4: This bit is the Even Parity Select bit. When bit 3 is a logic 1 and bit 4 is a logic 0, an odd number of logic 1s is transmitted or checked in the data word bits and Parity bit. When bit 3 is a logic 1 and bit 4 is a logic 1, an even number of logic 1s is transmitted or checked.

Bit 5: This bit is the Stick Parity bit. When bits 3, 4 and 5 are logic 1 the Parity bit is transmitted and checked as a logic 0. If bits 3 and 5 are 1 and bit 4 is a logic 0 then the Parity bit is transmitted and checked as a logic 1. If bit 5 is a logic 0 Stick Parity is disabled.

Bit 6: This bit is the Break Control bit. It causes a break condition to be transmitted to the receiving UART. When it is set to a logic 1, the serial output (SOUT) is forced to the Spacing (logic 0) state. The break is disabled by setting bit 6 to a logic 0. The Break Control bit acts only on SOUT and has no effect on the transmitter logic.

Note: This feature enables the CPU to alert a terminal in a computer communications system. If the following sequence is followed, no erroneous or extraneous characters will be transmitted because of the break.

1. Load an all 0s, pad character, in response to THRE.
2. Set break after the next THRE.
3. Wait for the transmitter to be idle. (TEMT = 1), and clear break when normal transmission has to be restored.

During the break, the Transmitter can be used as a character timer to accurately establish the break duration.

TABLE III. Baud Rates, Divisors and Crystals

Baud Rate	1.8432 MHz Cystal		3.072 MHz Crystal		18.432 MHz Crystal	
	Decimal Divisor for 16 × Clock	Percent Error	Decimal Divisor for 16 × Clock	Percent Error	Decimal Divisor for 16 × Clock	Percent Error
50	2304	—	3840	—	23040	—
75	1536	—	2560	—	15360	—
110	1047	0.026	1745	0.026	10473	—
134.5	857	0.058	1428	0.034	8565	—
150	768	—	1280	—	7680	—
300	384	—	640	—	3840	—
600	192	—	320	—	1920	—
1200	96	—	160	—	920	—
1800	64	—	107	0.312	640	—
2000	58	0.69	96	—	576	—
2400	48	—	80	—	480	—
3600	32	—	53	0.628	320	—
4800	24	—	40	—	240	—
7200	16	—	27	1.23	160	—
9600	12	—	20	—	120	—
19200	6	—	10	—	60	—
38400	3	—	5	—	30	—
56000	2	2.86	—	—	21	2.04
128000	—	—	—	—	9	—

Note: For baud rates of 250k, 300k, 375k, 500k, 750k and 1.5M using a 24 MHz crystal causes minimal error

TABLE II. Summary of Registers

Bit No.	Register Address											
	0 DLAB = 0	0 DLAB = 0	1 DLAB = 0	2	2	3	4	5	6	7	0 DLAB = 1	1 DLAB = 1
	Receiver Buffer Register (Read Only)	Transmitter Holding Register (Write Only)	Interrupt Enable Register	Interrupt Ident. Register (Read Only)	FIFO Control Register (Write Only)	Line Control Register	MODEM Control Register	Line Status Register	MODEM Status Register	Scratch Register	Divisor Latch (LS)	Divisor Latch (MS)
	RBR	THR	IER	IIR	FCR	LCR	MCR	LSR	MSR	SCR	DLL	DLM
0	Data Bit 0 (Note 1)	Data Bit 0	Enable Received Data Available Interrupt (ERBFI)	"0" if Interrupt Pending	FIFO Enable	Word Length Select Bit 0 (WLS0)	Data Terminal Ready (DTR)	Data Ready (DR)	Delta Clear to Send (DCTS)	Bit 0	Bit 0	Bit 8
1	Data Bit 1	Data Bit 1	Enable Transmitter Holding Register Empty Interrupt (ETBEI)	Interrupt ID Bit (0)	RCVR FIFO Reset	Word Length Select Bit 1 (WLS1)	Request to Send (RTS)	Overrun Error (OE)	Delta Data Set Ready (DDSR)	Bit 1	Bit 1	Bit 9
2	Data Bit 2	Data Bit 2	Enable Receiver Line Status Interrupt (ELSI)	Interrupt ID Bit (1)	XMIT FIFO Reset	Number of Stop Bits (STB)	Out 1	Parity Error (PE)	Trailing Edge Ring Indicator (TERI)	Bit 2	Bit 2	Bit 10
3	Data Bit 3	Data Bit 3	Enable MODEM Status Interrupt (EDSSI)	Interrupt ID Bit (2) (Note 2)	DMA Mode Select	Parity Enable (PEN)	Out 2	Framing Error (FE)	Delta Data Carrier Detect (DDCD)	Bit 3	Bit 3	Bit 11
4	Data Bit 4	Data Bit 4	0	0	Reserved	Even Parity Select (EPS)	Loop	Break Interrupt (BI)	Clear to Send (CTS)	Bit 4	Bit 4	Bit 12
5	Data Bit 5	Data Bit 5	0	0	Reserved	Stick Parity	0	Transmitter Holding Register (THRE)	Data Set Ready (DSR)	Bit 5	Bit 5	Bit 13
6	Data Bit 6	Data Bit 6	0	FIFOs Enabled (Note 2)	RCVR Trigger (LSB)	Set Break	0	Transmitter Empty (TEMT)	Ring Indicator (RI)	Bit 6	Bit 6	Bit 14
7	Data Bit 7	Data Bit 7	0	FIFOs Enabled (Note 2)	RCVR Trigger (MSB)	Divisor Latch Access Bit (DLAB)	0	Error in RCVR FIFO (Note 2)	Data Carrier Detect (DCD)	Bit 7	Bit 7	Bit 15

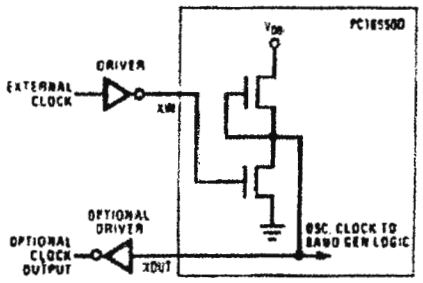
Note 1: Bit 0 is the least significant bit. It is the first bit serially transmitted or received

Note 2: These bits are always 0 in the 16450 Mode

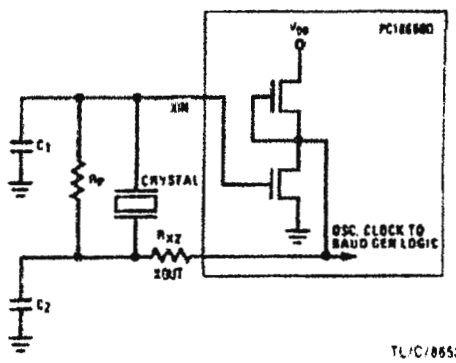
8.0 Registers (Continued)

Bit 7: This bit is the Divisor Latch Access Bit (DLAB). It must be set high (logic 1) to access the Divisor Latches of the Baud Generator during a Read or Write operation. It must be set low (logic 0) to access the Receiver Buffer, the Transmitter Holding Register, or the Interrupt Enable Register.

8.2 TYPICAL CLOCK CIRCUITS



TL/C/8652-19



TL/C/8652-20

Typical Crystal Oscillator Network (Note)

CRYSTAL	R_P	R_{X2}	C_1	C_2
3.1 MHz	1 MΩ	1.5k	10-30 pF	40-60 pF
1.8 MHz	1 MΩ	1.5k	10-30 pF	40-60 pF

Note: These R and C values are approximate and may vary 2x depending on the crystal characteristics. All crystal circuits should be designed specifically for the system

8.3 PROGRAMMABLE BAUD GENERATOR

The UART contains a programmable Baud Generator that is capable of taking any clock input from DC to 24 MHz and dividing it by any divisor from 2 to $2^{16}-1$. The output frequency of the Baud Generator is 16 × the Baud [divisor # = (frequency input) ÷ (baud rate × 16)]. Two 8-bit latches store the divisor in a 16-bit binary format. These Divisor Latches must be loaded during initialization to ensure proper operation of the Baud Generator. Upon loading either of the Divisor Latches, a 16-bit Baud counter is immediately loaded.

Table III provides decimal divisors to use with crystal frequencies of 1.8432 MHz, 3.072 MHz and 18.432 MHz, respectively. For baud rates of 38400 and below, the error obtained is minimal. The accuracy of the desired baud rate is dependent on the crystal frequency chosen. Using a divisor of zero is not recommended.

8.4 LINE STATUS REGISTER

This register provides status information to the CPU concerning the data transfer. Table II shows the contents of the Line Status Register. Details on each bit follow.

Bit 0: This bit is the receiver Data Ready (DR) indicator. Bit 0 is set to a logic 1 whenever a complete incoming character has been received and transferred into the Receiver Buffer Register or the FIFO. Bit 0 is reset to a logic 0 by reading all of the data in the Receiver Buffer Register or the FIFO.

Bit 1: This bit is the Overrun Error (OE) indicator. Bit 1 indicates that data in the Receiver Buffer Register was not read by the CPU before the next character was transferred into the Receiver Buffer Register, thereby destroying the previous character. The OE indicator is set to a logic 1 upon detection of an overrun condition and reset whenever the CPU reads the contents of the Line Status Register. If the FIFO mode data continues to fill the FIFO beyond the trigger level, an overrun error will occur only after the FIFO is full and the next character has been completely received in the shift register. OE is indicated to the CPU as soon as it happens. The character in the shift register is overwritten, but it is not transferred to the FIFO.

Bit 2: This bit is the Parity Error (PE) indicator. Bit 2 indicates that the received data character does not have the correct even or odd parity, as selected by the even-parity-select bit. The PE bit is set to a logic 1 upon detection of a parity error and is reset to a logic 0 whenever the CPU reads the contents of the Line Status Register. In the FIFO mode this error is associated with the particular character in the FIFO it applies to. This error is revealed to the CPU when its associated character is at the top of the FIFO.

Bit 3: This bit is the Framing Error (FE) indicator. Bit 3 indicates that the received character did not have a valid Stop bit. Bit 3 is set to a logic 1 whenever the Stop bit following the last data bit or parity bit is detected as a logic 0 bit (Spacing level). The FE indicator is reset whenever the CPU reads the contents of the Line Status Register. In the FIFO mode this error is associated with the particular character in the FIFO it applies to. This error is revealed to the CPU when its associated character is at the top of the FIFO. The UART will try to resynchronize after a framing error. To do this it assumes that the framing error was due to the next start bit, so it samples this "start" bit twice and then takes in the "data".

Bit 4: This bit is the Break Interrupt (BI) indicator. Bit 4 is set to a logic 1 whenever the received data input is held in the Spacing (logic 0) state for longer than a full word transmission time (that is, the total time of Start bit + data bits + Parity + Stop bits). The BI indicator is reset whenever the CPU reads the contents of the Line Status Register. In the FIFO mode this error is associated with the particular character in the FIFO it applies to. This error is revealed to the CPU when its associated character is at the top of the FIFO. When break occurs only one zero character is loaded into the FIFO. The next character transfer is enabled after SIN goes to the marking state and receives the next valid start bit.

Note: Bits 1 through 4 are the error conditions that produce a Receiver Line Status interrupt whenever any of the corresponding conditions are detected and the interrupt is enabled

8.0 Registers (Continued)

TABLE IV. Interrupt Control Functions

FIFO Mode Only	Interrupt Identification Register			Interrupt Set and Reset Functions			
Bit 3	Bit 2	Bit 1	Bit 0	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	0	0	1	—	None	None	—
0	1	1	0	Highest	Receiver Line Status	Overrun Error or Parity Error or Framing Error or Break Interrupt	Reading the Line Status Register
0	1	0	0	Second	Received Data Available	Receiver Data Available or Trigger Level Reached	Reading the Receiver Buffer Register or the FIFO Drops Below the Trigger Level
1	1	0	0	Second	Character Timeout Indication	No Characters Have Been Removed From or Input to the RCVR FIFO During the Last 4 Char. Times and There Is at Least 1 Char. in It During This Time	Reading the Receiver Buffer Register
0	0	1	0	Third	Transmitter Holding Register Empty	Transmitter Holding Register Empty	Reading the IIR Register (if source of interrupt) or Writing into the Transmitter Holding Register
0	0	0	0	Fourth	MODEM Status	Clear to Send or Data Set Ready or Ring Indicator or Data Carrier Detect	Reading the MODEM Status Register

Bit 5: This bit is the Transmitter Holding Register Empty (THRE) indicator. Bit 5 indicates that the UART is ready to accept a new character for transmission. In addition, this bit causes the UART to issue an interrupt to the CPU when the Transmit Holding Register Empty Interrupt enable is set high. The THRE bit is set to a logic 1 when a character is transferred from the Transmitter Holding Register into the Transmitter Shift Register. The bit is reset to logic 0 concurrently with the loading of the Transmitter Holding Register by the CPU. In the FIFO mode this bit is set when the XMIT FIFO is empty; it is cleared when at least 1 byte is written to the XMIT FIFO.

Bit 6: This bit is the Transmitter Empty (TEMT) indicator. Bit 6 is set to a logic 1 whenever the Transmitter Holding Register (THR) and the Transmitter Shift Register (TSR) are both empty. It is reset to a logic 0 whenever either the THR or TSR contains a data character. In the FIFO mode this bit is set to one whenever the transmitter FIFO and shift register are both empty.

Bit 7: In the 16450 Mode this is a 0. In the FIFO mode LSR7 is set when there is at least one parity error, framing error or break indication in the FIFO. LSR7 is cleared when the CPU reads the LSR, if there are no subsequent errors in the FIFO.

Note: The Line Status Register is intended for read operations only. Writing to this register is not recommended as this operation is only used for factory testing. In the FIFO mode the software must load a data byte in the Rx FIFO via Loopback Mode in order to write to LSR2–LSR4. LSR0 and LSR7 can't be written to in FIFO mode.

8.5 FIFO CONTROL REGISTER

This is a write only register at the same location as the IIR (the IIR is a read only register). This register is used to enable the FIFOs, clear the FIFOs, set the RCVR FIFO trigger level, and select the type of DMA signalling.

Bit 0: Writing a 1 to FCR0 enables both the XMIT and RCVR FIFOs. Resetting FCR0 will clear all bytes in both FIFOs. When changing from the FIFO Mode to the 16450 Mode and vice versa, data is automatically cleared from the FIFOs. This bit must be a 1 when other FCR bits are written to or they will not be programmed.

Bit 1: Writing a 1 to FCR1 clears all bytes in the RCVR FIFO and resets its counter logic to 0. The shift register is not cleared. The 1 that is written to this bit position is self-clearing.

Bit 2: Writing a 1 to FCR2 clears all bytes in the XMIT FIFO and resets its counter logic to 0. The shift register is not cleared. The 1 that is written to this bit position is self-clearing.

Bit 3: Setting FCR3 to a 1 will cause the RXRDY and TXRDY pins to change from mode 0 to mode 1 if FCR0 = 1 (see description of RXRDY and TXRDY pins).

Bit 4, 5: FCR4 to FCR5 are reserved for future use.

Bit 6, 7: FCR6 and FCR7 are used to set the trigger level for the RCVR FIFO interrupt.

7	6	RCVR FIFO Trigger Level (Bytes)
0	0	01
0	1	04
1	0	08
1	1	14

8.6 INTERRUPT IDENTIFICATION REGISTER

In order to provide minimum software overhead during data character transfers, the UART prioritizes interrupts into four levels and records these in the interrupt Identification Register. The four levels of interrupt conditions in order of priority are Receiver Line Status; Received Data Ready; Transmitter Holding Register Empty; and MODEM Status.

8.0 Registers (Continued)

When the CPU accesses the IIR, the UART freezes all interrupts and indicates the highest priority pending interrupt to the CPU. While this CPU access is occurring, the UART records new interrupts, but does not change its current indication until the access is complete. Table II shows the contents of the IIR. Details on each bit follow:

Bit 0: This bit can be used in a prioritized interrupt environment to indicate whether an interrupt is pending. When bit 0 is a logic 0, an interrupt is pending and the IIR contents may be used as a pointer to the appropriate interrupt service routine. When bit 0 is a logic 1, no interrupt is pending.

Bits 1 and 2: These two bits of the IIR are used to identify the highest priority interrupt pending as indicated in Table IV.

Bit 3: In the 16450 Mode this bit is 0. In the FIFO mode this bit is set along with bit 2 when a timeout interrupt is pending.

Bits 4 and 5: These two bits of the IIR are always logic 0.

Bits 6 and 7: These two bits are set when FCR0 = 1.

8.7 INTERRUPT ENABLE REGISTER

This register enables the five types of UART interrupts. Each interrupt can individually activate the interrupt (INTR) output signal. It is possible to totally disable the interrupt system by resetting bits 0 through 3 of the Interrupt Enable Register (IER). Similarly, setting bits of the IER register to a logic 1, enables the selected interrupt(s). Disabling an interrupt prevents it from being indicated as active in the IIR and from activating the INTR output signal. All other system functions operate in their normal manner, including the setting of the Line Status and MODEM Status Registers. Table II shows the contents of the IER. Details on each bit follow.

Bit 0: This bit enables the Received Data Available Interrupt (and timeout interrupts in the FIFO mode) when set to logic 1.

Bit 1: This bit enables the Transmitter Holding Register Empty interrupt when set to logic 1.

Bit 2: This bit enables the Receiver Line Status Interrupt when set to logic 1.

Bit 3: This bit enables the MODEM Status Interrupt when set to logic 1.

Bits 4 through 7: These four bits are always logic 0.

8.8 MODEM CONTROL REGISTER

This register controls the interface with the MODEM or data set (or a peripheral device emulating a MODEM). The contents of the MODEM Control Register are indicated in Table II and are described below.

Bit 0: This bit controls the Data Terminal Ready ($\overline{DTR}$) output. When bit 0 is set to a logic 1, the $\overline{DTR}$ output is forced to a logic 0. When bit 0 is reset to a logic 0, the $\overline{DTR}$ output is forced to a logic 1.

Note: The $\overline{DTR}$ output of the UART may be applied to an EIA inverting line driver (such as the DS1488) to obtain the proper polarity input at the succeeding MODEM or data set.

Bit 1: This bit controls the Request to Send ($\overline{RTS}$) output. Bit 1 affects the $\overline{RTS}$ output in a manner identical to that described above for bit 0.

Bit 2: This bit controls the Output 1 ($\overline{OUT\ 1}$) signal, which is an auxiliary user-designated output. Bit 2 affects the $\overline{OUT\ 1}$ output in a manner identical to that described above for bit 0.

Bit 3: This bit controls the Output 2 ($\overline{OUT\ 2}$) signal, which is an auxiliary user-designated output. Bit 3 affects the $\overline{OUT\ 2}$ output in a manner identical to that described above for bit 0.

Bit 4: This bit provides a local loopback feature for diagnostic testing of the UART. When bit 4 is set to logic 1, the following occur: the transmitter Serial Output (SOUT) is set to the Marking (logic 1) state; the receiver Serial Input (SIN) is disconnected; the output of the Transmitter Shift Register is "looped back" into the Receiver Shift Register input; the four MODEM Control inputs ($\overline{DSR}$, $\overline{CTS}$, $\overline{RI}$, and $\overline{DCD}$) are disconnected; and the four MODEM Control outputs ($\overline{DTR}$, $\overline{RTS}$, $\overline{OUT\ 1}$, and $\overline{OUT\ 2}$) are internally connected to the four MODEM Control inputs, and the MODEM Control output pins are forced to their inactive state (high). In the loopback mode, data that is transmitted is immediately received. This feature allows the processor to verify the transmit-and received-data paths of the UART.

In the loopback mode, the receiver and transmitter interrupts are fully operational. Their sources are external to the part. The MODEM Control interrupts are also operational, but the interrupts' sources are now the lower four bits of the MODEM Control Register instead of the four MODEM Control inputs. The interrupts are still controlled by the Interrupt Enable Register.

Bits 5 through 7: These bits are permanently set to logic 0.

8.9 MODEM STATUS REGISTER

This register provides the current state of the control lines from the MODEM (or peripheral device) to the CPU. In addition to this current-state information, four bits of the MODEM Status Register provide change information. These bits are set to a logic 1 whenever a control input from the MODEM changes state. They are reset to logic 0 whenever the CPU reads the MODEM Status Register.

The contents of the MODEM Status Register are indicated in Table II and described below.

Bit 0: This bit is the Delta Clear to Send (DCTS) indicator. Bit 0 indicates that the $\overline{CTS}$ input to the chip has changed state since the last time it was read by the CPU.

Bit 1: This bit is the Delta Data Set Ready (DDSR) indicator. Bit 1 indicates that the $\overline{DSR}$ input to the chip has changed state since the last time it was read by the CPU.

Bit 2: This bit is the Trailing Edge of Ring Indicator (TERI) detector. Bit 2 indicates that the $\overline{RI}$ input to the chip has changed from a low to a high state.

Bit 3: This bit is the Delta Data Carrier Detect (DDCD) indicator. Bit 3 indicates that the $\overline{DCD}$ input to the chip has changed state.

Note: Whenever bit 0, 1, 2, or 3 is set to logic 1, a MODEM Status Interrupt is generated.

Bit 4: This bit is the complement of the Clear to Send ($\overline{CTS}$) input. If bit 4 (loop) of the MCR is set to a 1, this bit is equivalent to RTS in the MCR.

Bit 5: This bit is the complement of the Data Set Ready ($\overline{DSR}$) input. If bit 4 of the MCR is set to a 1, this bit is equivalent to DTR in the MCR.

Bit 6: This bit is the complement of the Ring Indicator ($\overline{RI}$) input. If bit 4 of the MCR is set to a 1, this bit is equivalent to OUT 1 in the MCR.

8.0 Registers (Continued)

Bit 7: This bit is the complement of the Data Carrier Detect ($\overline{DCD}$) input. If bit 4 of the MCR is set to a 1, this bit is equivalent to OUT 2 in the MCR.

8.10 SCRATCHPAD REGISTER

This 8-bit Read/Write Register does not control the UART in anyway. It is intended as a scratchpad register to be used by the programmer to hold data temporarily.

8.11 FIFO INTERRUPT MODE OPERATION

When the RCVR FIFO and receiver interrupts are enabled (FCR0 = 1, IER0 = 1) RCVR interrupts will occur as follows:

A. The receive data available interrupt will be issued to the CPU when the FIFO has reached its programmed trigger level; it will be cleared as soon as the FIFO drops below its programmed trigger level.

B. The IIR receive data available indication also occurs when the FIFO trigger level is reached, and like the interrupt it is cleared when the FIFO drops below the trigger level.

C. The receiver line status interrupt (IIR = 06), as before, has higher priority than the received data available (IIR = 04) interrupt.

D. The data ready bit (LSR0) is set as soon as a character is transferred from the shift register to the RCVR FIFO. It is reset when the FIFO is empty.

When RCVR FIFO and receiver interrupts are enabled, RCVR FIFO timeout interrupts will occur as follows:

A. A FIFO timeout interrupt will occur, if the following conditions exist:

- — at least one character is in the FIFO
- — the most recent serial character received was longer than 4 continuous character times ago (if 2 stop bits are programmed the second one is included in this time delay).
- — the most recent CPU read of the FIFO was longer than 4 continuous character times ago.

The maximum time between a received character and a timeout interrupt will be 160 ms at 300 baud with a 12-bit receive character (i.e., 1 Start, 8 Data, 1 Parity and 2 Stop Bits).

B. Character times are calculated by using the RCLK input for a clock signal (this makes the delay proportional to the baudrate).

C. When a timeout interrupt has occurred it is cleared and the timer reset when the CPU reads one character from the RCVR FIFO.

D. When a timeout interrupt has not occurred the timeout timer is reset after a new character is received or after the CPU reads the RCVR FIFO.

When the XMIT FIFO and transmitter interrupts are enabled (FCR0 = 1, IER1 = 1), XMIT interrupts will occur as follows:

A. The transmitter holding register interrupt (02) occurs when the XMIT FIFO is empty; it is cleared as soon as the transmitter holding register is written to (1 to 16 characters may be written to the XMIT FIFO while servicing this interrupt) or the IIR is read.

B. The transmitter FIFO empty indications will be delayed 1 character time minus the last stop bit time whenever the following occurs: THRE = 1 and there have not been at least two bytes at the same time in the transmit FIFO. Since the last THRE = 1. The first transmitter interrupt after changing FCR0 will be immediate, if it is enabled.

Character timeout and RCVR FIFO trigger level interrupts have the same priority as the current received data available interrupt; XMIT FIFO empty has the same priority as the current transmitter holding register empty interrupt.

8.12 FIFO POLLED MODE OPERATION

With FCR0 = 1 resetting IER0, IER1, IER2, IER3 or all to zero puts the UART in the FIFO Polled Mode of operation. Since the RCVR and XMITTER are controlled separately either one or both can be in the polled mode of operation.

In this mode the user's program will check RCVR and XMITTER status via the LSR. As stated previously:

LSR0 will be set as long as there is one byte in the RCVR FIFO

LSR1 to LSR4 will specify which error(s) has occurred. Character error status is handled the same way as when in the interrupt mode, the IIR is not affected since IER2 = 0

LSR5 will indicate when the XMIT FIFO is empty.

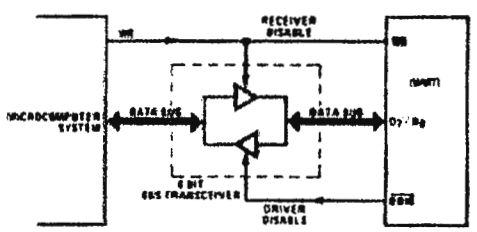
LSR6 will indicate that both the XMIT FIFO and shift register are empty.

LSR7 will indicate whether there are any errors in the RCVR FIFO.

There is no trigger level reached or timeout condition indicated in the FIFO Polled Mode, however, the RCVR and XMIT FIFOs are still fully capable of holding characters.

9.0 Typical Applications

Typical Interface for a High-Capacity Data Bus

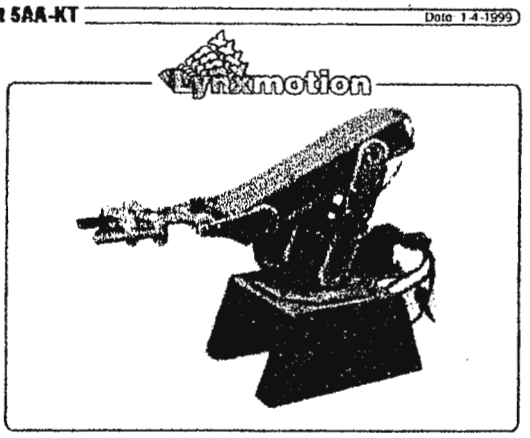


TL/C/8652-23

ANEXO D.
CARACTERISTICAS DEL BRAZO.

The 5 Axis Robotic Arm Kit

- *Pre-assembled electronics.*
- *Easy to assemble kit means you are up and running in only a few hours!*
- *Position in an X, Y, Z grid with the new RoboMotion for Windows software.*
- *Excellent teaching aid or science fair project.*
- *Mobile version also available.*
- *Write your own program with any language capable of sending serial data.*
- *Easy to control from a PC or microcontroller with simple 3 byte command for movement.*
- *Intermediate experience level kit.*



5 Axis Robotic Arm Information

The Lynxmotion 5 Axis Robotic Arm Kit delivers fast, accurate and repeatable movement. The robot features; base rotation, shoulder, elbow and wrist motion, with a functional gripper to make five independent axis of movement. No soldering is required for the electronics. With the exception of some basic construction supplies, all of the components are included to assemble a functional robot. A host PC or microcontroller is required to issue simple positioning commands for movement. The Lynxmotion 5 Axis Robotic Arm Kit is a very affordable introductory system. Some of the uses include: hobby robotics, tech school senior project, science fair project, light industrial proof of concept, technical education, artificial intelligence programming and experimentation, etc.

Specifications

- Axis (including gripper) = 5
- Motion control (servos) = closed loop (local)
- Height (home position) = 7"
- Height (reaching up) = 15"
- Reach (forward) = 9.5"
- Gripper opening = 1.25"
- Lift weight (arm extended) = approx. 3 oz
- Weight (without batteries) = 21.3 oz
- Range of motion per axis = 90° or 180° switch selectable
- Accuracy of motion per axis = .36° or .72° switch selectable
- Servo voltage = rated for 6 vdc, commonly used at 7.2 vdc
- Servo current required (idle) = 10mA each
- Servo current required (moving) = 130mA each
- Servo torque = 49 oz-in
- Servo speed = 90° in .24 Sec. (can be slowed to prevent backlash)
- SSC current requirements = 4.7mA

The kit includes

Electronics

SSC (Serial Servo Controller) Easily control up to 8 servos with this pre-assembled unit from a PC, Basic Stamp, or other microcontroller.

Mechanics

All the hardware and structural components to build the robot arm, base and gripper, and six high quality Hitec servo motors.

Software

Floppy disks for IBM-PC with DOS and Windows programs. The DOS software is written in Quick BASIC ver 4.5. It allows you to move the arm via the keyboard, save positions to a script file, single step and play the scripts back, save and load the scripts to disk. The source code is included in ascii so you can modify it with any basic. The Windows program allows you to teach the robot from the keyboard or joy stick. The arms gripper is positioned in an X, Y, Z grid in inches, and the moves are stored in a spreadsheet format to facilitate easy editing.

Illustrated Documentation

- Robot Assembly Manual 5AA-0-7 Ver 6.0
- Using the SSC

Work Envelope

Figure 1 below illustrates the side view of the work envelope. This is the area that the gripper may be positioned in for doing its task. This diagram assumes the arm is using the SSC's extended range of 180°, and would be slightly smaller using the 90° (normal) range.

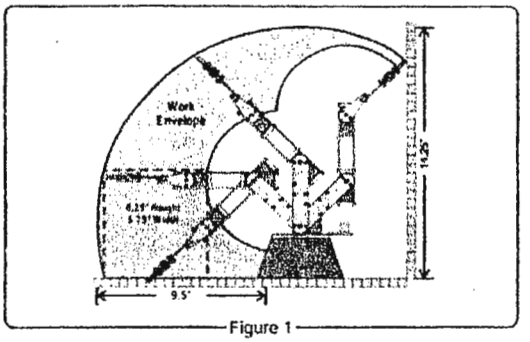


Figure 1

Ordering Information

To make an operational arm, you will need the 5AA-KT and the DB9-01 serial data cable for the PC. The servos require 4.8 to 7.2 vdc at 1.5 amps. We recommend batteries, but a wall pack should work fine. The Windows software is an advanced program and is recommended for doing complex tasks. The MRA-KT makes the arm mobile, but additional sensors and microcontrollers will be required to make it autonomous. The Basic Stamp 2 can be used to control the arm. This approach allows the additional I/O to be used in more complex experiments, but requires user programming.

Part No.	Description	Price
5AA-KT	5 Axis Robotic Arm Kit	$195.00
DB9-01	DB9 Serial Data Cable for PC	$6.00
RM-01	RoboMotion Windows Software	$30.00
TOH-01	Towers of Hanoi Puzzle	$20.00
	-Servo cable adapters for easier assembly -	
SEA-01	Servo 6" Ext. Adapter (1 required)	$4.90
SYA-01	Servo "Y" Adapter (1 required)	$8.75
	-Can be made to be Mobile -	
M5AA-KT	Mobile 5AA-KT Kit	$250.00
	-Can be controlled by the Basic Stamp 2 -	
BS2-01	Basic Stamp 2 IC	$49.00
BS2C-01	BS2 Carrier	$20.00
BSP-01	Basic Stamp Programming Pack	$99.00
	-Other useful documentation -	
BSB-01	Prog. & Cust. the Basic Stamp Comp.	$35.00
MRB-01	Mobile Robots Book Second Edition	$32.00

Note: BASIC Stamp and PBASIC are trademarks of Parallax Inc.

Lynxmotion Robot Kits - Tel:309-382-1816 - Fax:309-382-1254 - http://www.lynxmotion.com - Copyright, Lynxmotion, Inc. 1999

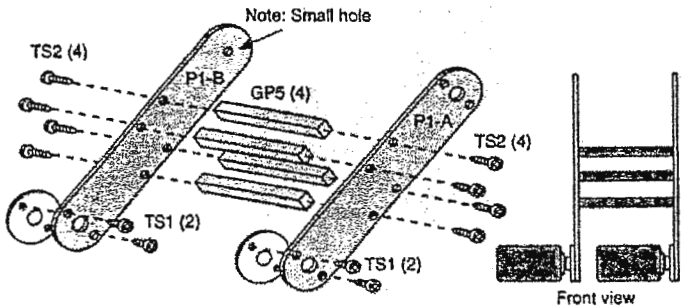
Shoulder Assembly (Do This First)

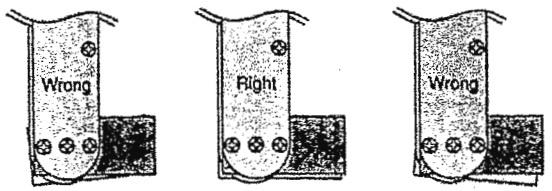
The shoulder servos are connected in parallel both electrically and mechanically. The servos are not precision devices, so you need to find two servos that have matching output shaft positions for use in the shoulder. Failure to do this will result in the servos fighting each other and drawing excess current.

1) Put together the shoulder assembly as shown. Remove the round servo horns from all of the servos. Note, the round servo horns attach from the same side of the assembly.

2) Attach two servos to the SSC and power them up. The servos should zip to the mid position. Temporarily push them onto the assembly as shown below. Look at the assembly from the side to check the alignment of the servos. If they do not line up replace one of the servos with another. Keep comparing the alignment until you find the best matching pair. Leave the servos connected to the assembly and set it aside for now.

Note: It will be necessary to control both shoulder servos from the same channel on the SSC. You can purchase a servo cable "Y" adapter from your local hobby shop, or you can cut and splice the servo wires. Wait until the robot is assembled and the SSC is mounted before cutting and splicing wires to ensure the wire is long enough.





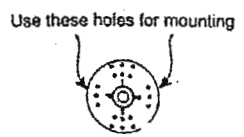
Note: With the servos connected to the SSC and power applied, view the assembly from this angle to ensure the two shoulder servos lign up properly.

You will need;

TS1 (4)

TS2 (8)

P1-A, P1-B, GP5 (4)



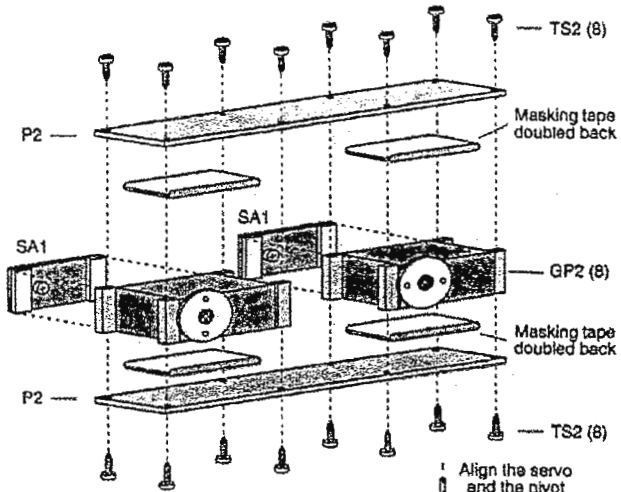
Forearm Assembly

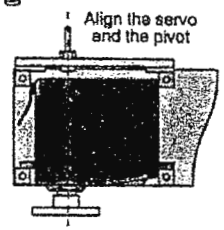
1) Install the 8 crossamembers on the bottom plate, using the self tapping screws.

2) Mount the servos in place as shown below. In order to keep the servos from moving around you can use some masking tape doubled back with the sticky side out.

3) Install the top panel.

4) Assemble the pivot sub assemblies and attach them to the arm as shown.





= cyanoacrylate or threadlock

You will need;

TS2 (16)

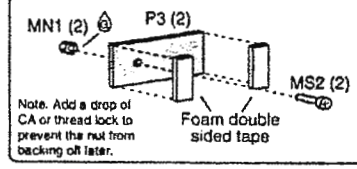
MS2 (2)

MN1 (2)

P2 (2)

GP2 (8)

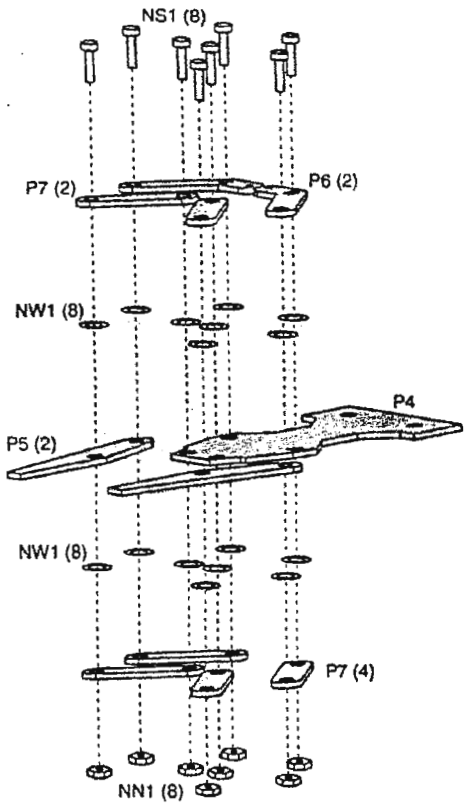
SA1 pivot sub-assembly 2 required



Gripper Assembly

1) Assemble the gripper as shown. Make sure the nylon hardware is completely free spinning. If the hardware has even a little friction, the gripper may not open properly.

2) You may want to add a thread lock to the nuts to prevent them from backing off while in use.



You will need;

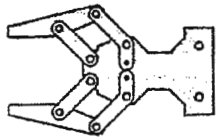
NS1 (8)

NW1 (16)

NN1 (8)

P4, P5 (2), P6 (2), P7 (6)

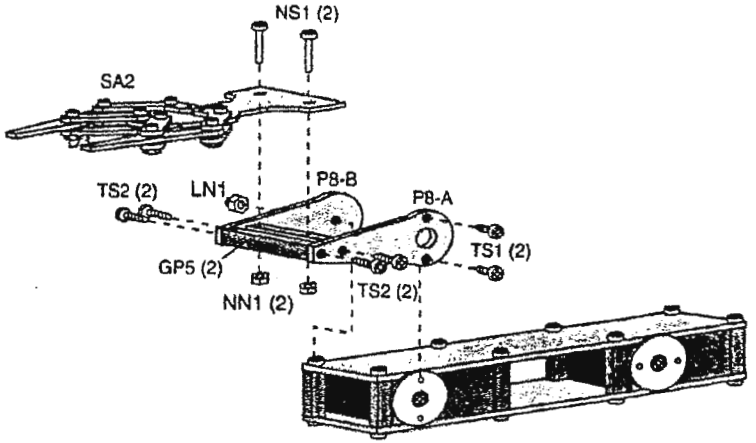
SA2 gripper sub assembly top view



Wrist Assembly

1) Build the wrist assembly as shown below.

2) Attach the gripper to the wrist assembly and then connect the wrist assembly to the forearm.



You will need;

TS1 (2)

TS2 (4)

NS1 (2)

NN1 (2)

LN1

GP5 (2), P8-A, P8-B

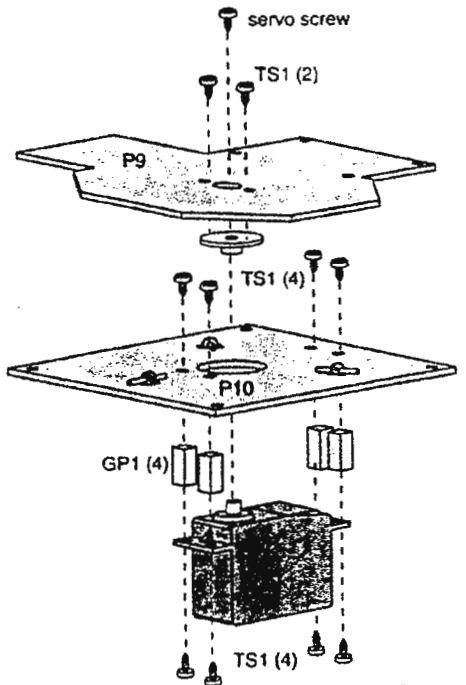
Rotator Assembly

1) Attach the servo mounting crossmembers to the robot base, caution don't overtighten. Install the servo to the servo mounting crossmembers, again be careful not to overtighten.

2) Glue the roller bearing assemblies to the base as shown. To prevent squeeking apply a drop of oil to each bearing.

3) Connect the round servo horn to the rotator base.

4) Ensure the servo is in mid position and install the rotator base with the pan handle pointing to the left. The rotator base should turn freely and be supported by the rollers not the servo. The rotator base should be flat, not bowed in at the center. If it seems to be a little stiff loosen the servo screw a bit and pull the rotator base up away from the servo.



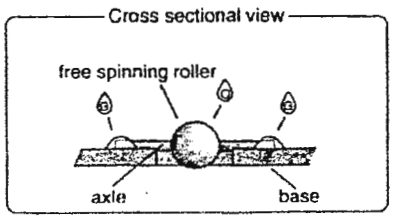
= oil

= cyanoacrylate glue

You will need;

TS1 (10)

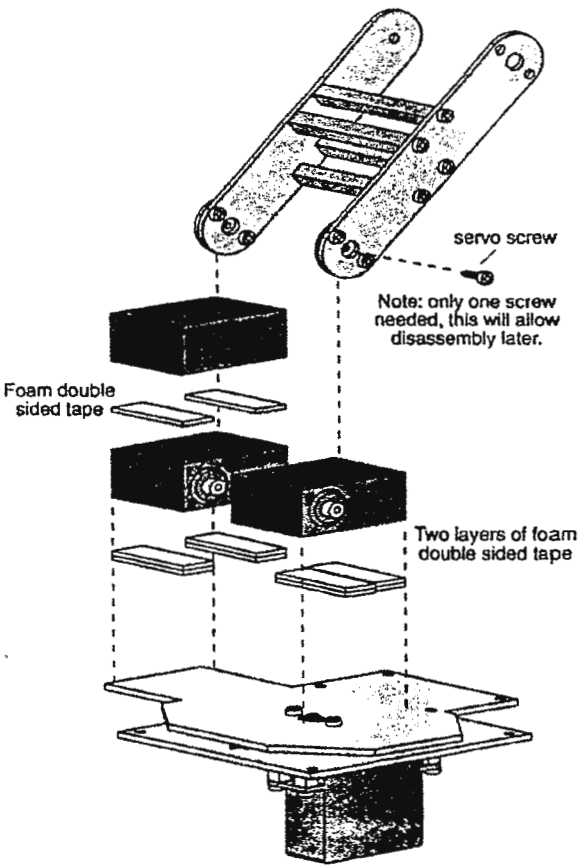
GP1 (4)
Rollers / Axles



Shoulder Servo Assembly

1) Apply two layers of the foam double sided tape to the two shoulder servos. The outside servo has the tape oriented to the front and rear of the servo. The inside servo has the tape oriented to the rear of the servo. This is necessary to clear the heads of the screws on the rotator base.

2) Apply two more strips of the foam double sided tape to the top of the outside shoulder servo. Place the gripper servo onto the tape with the output shaft closer to the front of the base.



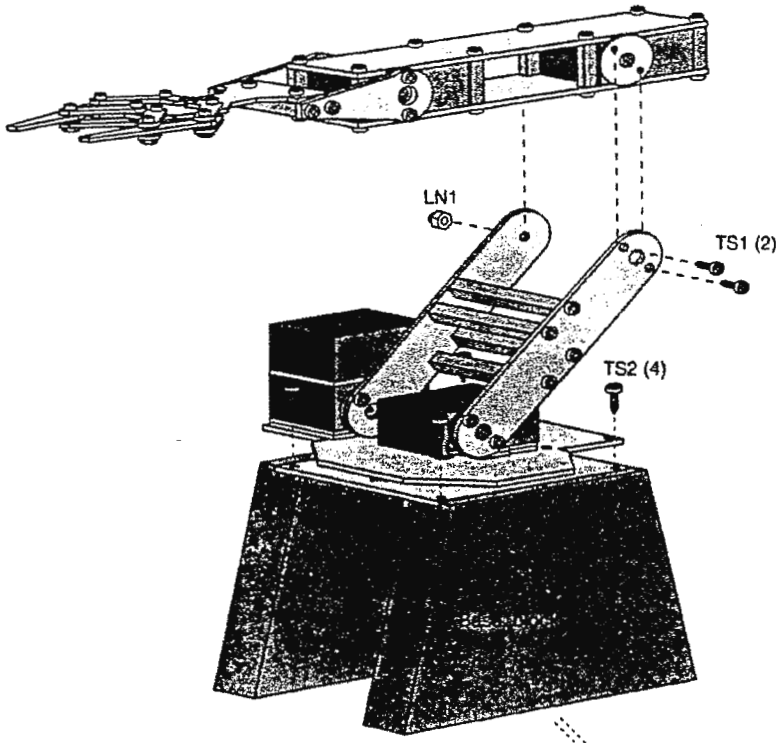
You will need;

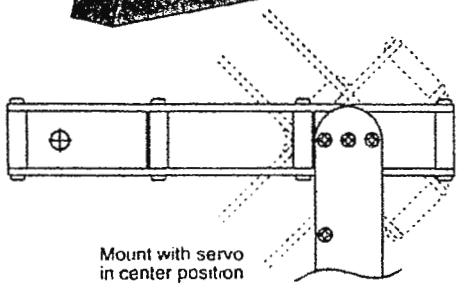
Foam double sided tape

Elbow Assembly

1) Ensure the elbo servo is in the center of rotation and attach the forearm assembly to the shoulder assembly as illustrated below.

2) If you are building the stationary arm use the two wood blocks to make the base. Don't forget to paint them flat black for a more attractive robot. Just use the TS2 screws to attach the arm to the wood blocks. If you are building the Mobile Arm then use the provided 4-40 machine screws to attach the arm to the mobile base.





You will need;

TS1 (2)

TS2 (4)

LN1

Cabling Assembly

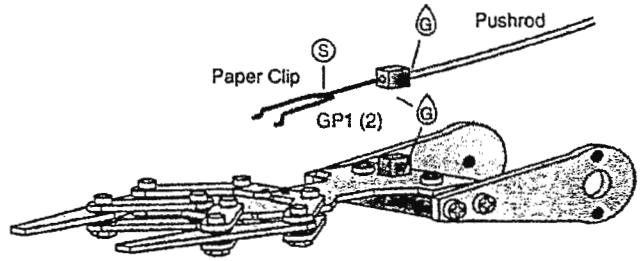
1) Remove the metal pushrod and cut the cable sheath (yellow plastic) to 16". Use a hobby knife with a wood block to cut a clean edge, as wire cutters can crimp the end making the cable stick.

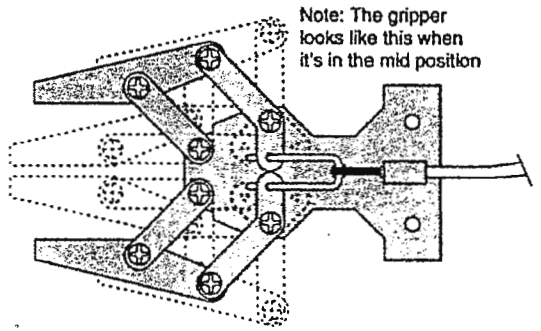
2) Glue the two parts GP1 to the wrist.

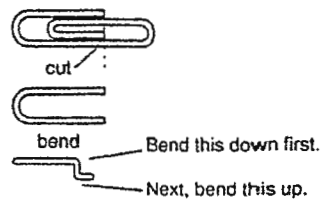
3) Glue the cable sheath to the center of the top GP1 piece. Do not get glue inside the sheath.

4) Reinsert the pushrod, cut and bend the paper clip wire as shown, and insert it into the lever arms.

5) Solder the pushrod to the paper clip.







Ⓢ = solder connection

Ⓖ = cyanoacrylate glue

You will need;

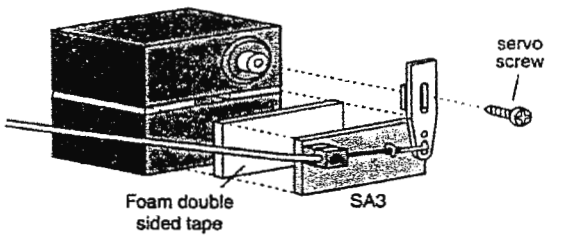
GP1 (2)
Paper clip, Pushrod

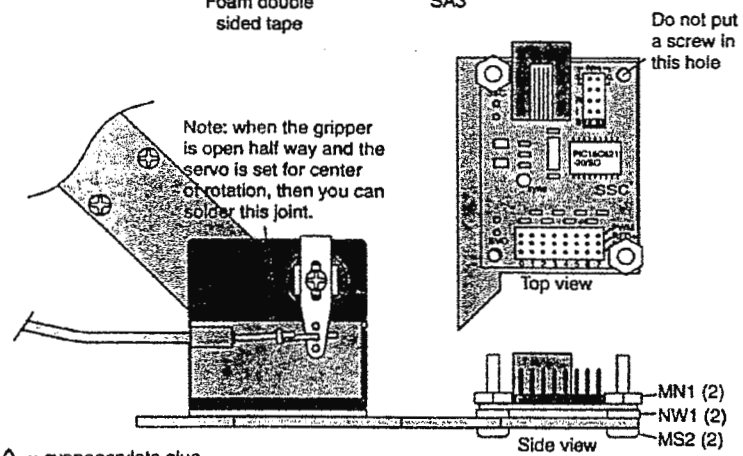
Cabling Assembly (cont.)

6) Glue the two GP1 parts to the plastic piece as offsets. Attach the cabling assembly to the servo with foam double sided tape.

7) Insert the cable sheath in the center of the top piece and glue it in place.

8) Cut and bend the remaining section of paper clip wire as shown and solder it to the pushrod of the cable, ensure the servo and the gripper are at mid position. You may need to experiment with the throw of the servo.



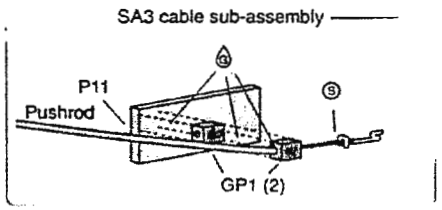


Ⓐ = cyanoacrylate glue

Ⓢ = solder connection

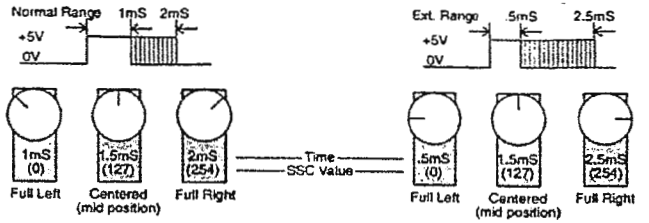
You will need:

MS2 (2)
MN1 (2)
NW1 (2)
P11
GP1 (2)
Paper Clip

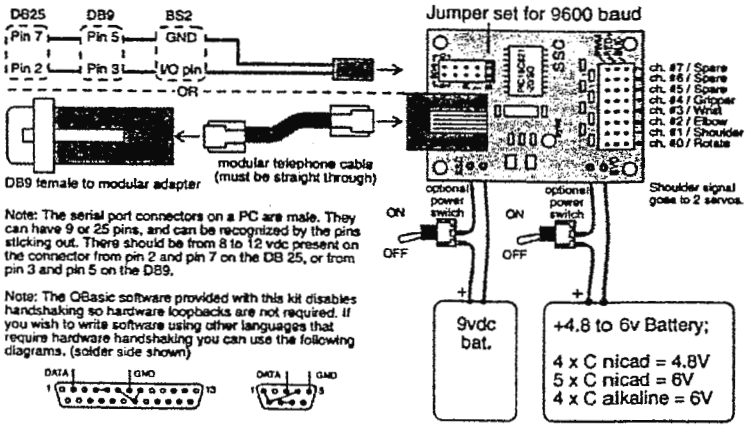


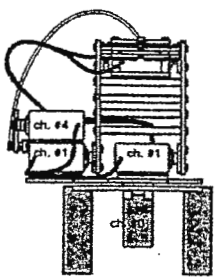
SCC Information

The following illustration shows the correlation between the SSC control value, the pulse length, and the actual servo output shaft position. This system is proportional so you may position the servo anywhere in between the points illustrated with a resolution of .36 ° normal range or .72° extended range. For specific SSC programming issues please consult the Using the Mini SSC II (Serial Servo Controller) manual.

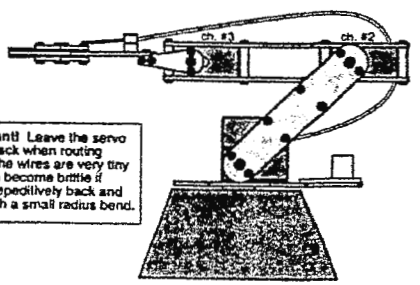


Wiring and Communication





Important! Leave the servo wires slack when routing them. The wires are very tiny and can become brittle if flexed repeditively back and forth with a small radius bend.



ANEXO F.
APLICACIÓN DEL PRINCIPIO DE MINIMOS CUADRADOS.

En muchos proyectos (de ingeniería especialmente) se requiere aplicar algún principio matemático (como la derivada o la integral) para realizar algún proceso, sin embrago, comúnmente no se dispone de las ecuaciones que rigen a los elementos prácticos de un experimento. Por fortuna, existe una manera de obtener expresiones matemáticas a partir de medidas experimentales.

El procedimiento en cuestión se basa en el principio estadístico de los mínimos cuadrados. Considerése el caso más sencillo de una recta que se ajuste a los valores medidos. Supongamos que se tiene un conjunto de N valores de una variable Y, medidos como función de la variable X. Debemos de restringirnos al caso especial de que toda la incertidumbre se limita a la variable Y, esto quiere decir que los valores de la X se conocen muy exactamente (o al menos con una mayor precisión) comparativamente con los valores de Y. El método puede ampliarse para abarcar el caso de incertidumbre en ambas dimensiones, pero el procedimiento no es muy sencillo.

La pregunta por contestar ahora en nuestro procedimiento matemático es: ¿Cuál de todas las líneas en el plano X-Y se escogerá como la mejor y en base a qué criterio? El principio de los mínimos cuadrados permite hacer esta selección en base a las desviaciones de los puntos en dirección vertical a partir de las líneas. Vea la figura que se muestra a continuación:

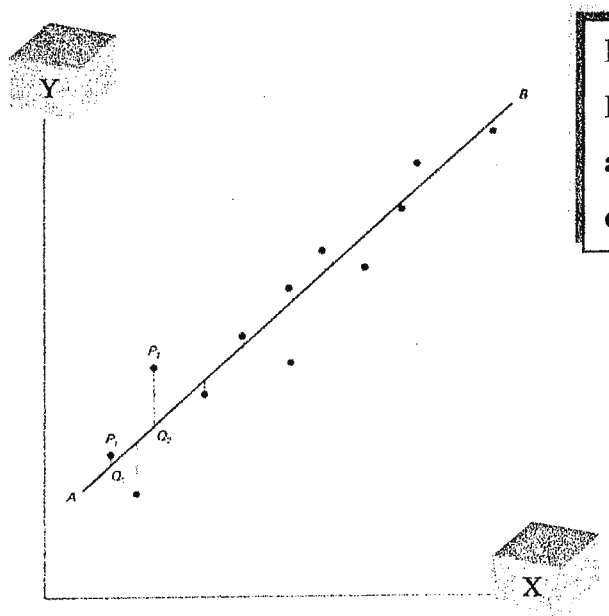


Fig. F-1.
Representación gráfica de la aproximación de ploteo experimental a una función.

Sea AB una candidata a la categoría de la "mejor" línea. Consideremos todos los intervalos verticales entre los puntos y la línea, de los cuales P_2O_2 es típico. Se define como mejor línea aquella que minimiza la suma de los cuadrados de las desviaciones como P_2O_2.

Nótese que no tenemos la oportunidad de considerar que un criterio inventado como éste proporcione algún camino automático a las respuestas "correctas". Se trata, simplemente, de una opción de criterio para optimizar la trayectoria, es decir, de una opción para optimizar la trayectoria de nuestra línea entre los puntos. Hay que reconocer, empero, que sí ofrece algunas ventajas sobre otras posibilidades, como minimizar la tercera potencia de los intervalos, o la primera, etc. Aunque no hace falta, en general, preocuparse directamente de la justificación lógica del principio de mínimos cuadrados tal como se usa aquí, es interesante darse cuenta de cuál es la base de su validez propuesta. Se puede probar que el procedimiento de minimizar los cuadrados de las desviaciones da lugar, en muestreos repetidos, a una menor varianza de los parámetros resultantes, como por ejemplo la pendiente, que al usar cualquier otro criterio. En consecuencia, se tiene derecho a confiar más en los resultados obtenidos usando el principio de mínimos cuadrados, que en el caso de cualquier otro método comparable; de aquí que el uso de este principio esté muy difundido.

Se expresará ahora el principio de mínimos cuadrados en forma matemática. Defináse que la mejor línea es aquella que lleva a su valor mínimo la suma:

$$\sum (P_i O_i)^2$$

y se desea obtener los parámetros, pendiente m y ordenada al origen b, de esa mejor línea.

Sea la ecuación de la mejor línea:

$$Y = m\,x + b$$

La magnitud de la desviación $P_i Q_i$ es el intervalo entre un cierto valor medido y_i, y el valor de y en ese punto, para el valor de x. Este valor y se puede calcular a partir del valor correspondiente de x como $mx_i + b$, de modo que si le llamase Δy_i a cada diferencia, se tendrá:

$$\Delta y_i = y_i - (m\,x_i + b)$$

El criterio de mínimos cuadrados permite obtener los valores deseados de m y b, a partir de la condición:

$$\sum [yi - (mxi + b)]^2 = \text{mínimo}$$

Que se puede escribir como:

$$\sum [yi - (mxi + b)]^2 = M$$

Luego, la condición para que sea un mínimo es:

$$\partial M / \partial m = 0 \qquad \text{y} \qquad \partial M / \partial b = 0$$

Un breve ejercicio algebráico permite entonces obtener los valores de la pendiente y la ordenada al origen de la mejor línea, que son:

$$m = \frac{N \Sigma (x_i y_i) - \Sigma x_i \Sigma y_i}{N \Sigma x_i^2 - (\Sigma x_i)^2}$$

$$b = \frac{\Sigma x_i^2 \Sigma y_i - \Sigma x_i \Sigma (x_i y_i)}{N \Sigma x_i^2 - (\Sigma x_i)^2}$$

Con esto, se ha logrado reemplazar el uso a veces cuestionable del juicio personal, por un procedimiento matemático, que da resultados muy precisos y de aceptabilidad universal. Además, como el nuevo método tiene cierta significación estadística, cabe esperar una forma más precisa para el cálculo de incertidumbre. De hecho, el principio de mínimos cuadrados permite obtener inmediatamente valores de la desviación estándar de la pendiente y la ordenada al origen, lo que

da incertidumbres de significación estadística conocida.

La desviación estándar de la pendiente y la ordenada al origen se calculan en términos de la desviación estándar de la distributión de valores de ∂y alrededor de la mejor linea, que se le llamará S_y. Esta última está dada por:

$$S_y = \sqrt{\frac{\Sigma\,(\delta y_i)^2}{N-2}}$$

No se preocupe porque una desviación estándar se calcule usando N - 2 en el denominador, en vez de N - 1 o N; esto es consecuencia de aplicar la definición de la desviación estándar a la posición de una linea en un plano. Los valores de S_m y S_b están dados entonces por:

$$S_m = S_y \times \sqrt{\frac{N}{N\,\Sigma\,x_i^2 - (\Sigma\,x_i)^2}}$$

$$S_b = S_y \times \sqrt{\frac{\Sigma\,x_i^2}{N\,\Sigma\,x_i^2 - (\Sigma\,x_i)^2}}$$

Estos pueden utilizarse en combinación con los valores de *m* y *b* para indicar intervalos con el significado normal, es decir que los intervalos de una desviación estándar dan una probabilidad del 68% de encerrar el valor central del universo en cuestión, dos desviaciones estándar 95%, etc. Una ventaja muy importante del método de mínimos cuadrados es que proporciona valores estadísticamente significativos de las incertidumbres en la pendiente y la ordenada al origen, que se derivan objetivamente de la dispersión real en los mismos puntos, sin perjuicio de cualquier afirmación optimista que se quiera hacer sobre las incertidumbres de los valores medidos.

Si, en el experimento, los puntos utilizados en el cálculo de mínimos cuadrados no son igualmente

precisos, se debe usar algún procedimiento que le asocie mayor importancia a las mediciones más precisas. Este procedimiento se denomina "ponderación". El uso de los procedimientos de ponderación no se limíta al ajuste de lineas rectas; son aplicables siempre que se quiera combinar observaciones de forma matemática, aún en el caso de un proceso tan sencillo como el de determinar la media de un conjunto de valores de precisión desigual. Las ecuaciones para encontrar una media ponderada y para hacer un cálculo ponderado de mínimos cuadrados se muestran a continuación.

Consideremos que tenemos un conjunto de cantidades x_i medidas independientes, y que conocemos como la desviación estándar, S_i , para cada una de las x_i. La media ponderada del conjunto de valores x está dada por:

$$\bar{x} = \frac{\sum \frac{x_i}{S_i^2}}{\sum \frac{1}{S_i^2}}$$

y la desviación estándar de la media ponderada por:

$$S^2 = \frac{\sum \frac{(x_i - \bar{x})^2}{S_i^2}}{(N-1)\sum \frac{1}{S_i^2}}$$

AJUSTE POR MINIMOS CUADRADOS DE FUNCIONES NO LINEALES

Los procedimientos empleados en el ejemplo anterior para determinar la pendiente y la ordenada al origen de la mejor línea recta pueden, desde luego, aplicarse, al menos en principio, a funciones no lineales. Es factible expresar una ecuación análoga a la ecuación de la punto pendiente para cualquier función, y todavia aplicar como requisisto que para expresar la obtención del mínimo de

la cantidad M con respecto a los parámetros del modelo, las derivadas con respecto a cada variable a buscar deben ser cero. Si las ecuaciones que resultan para los parámetros son fáciles de resolver, se puede proceder a calcular sus valores tal como se hizo para las líneas rectas.

Con frecuencia, sin embargo, no es fácil resolver las ecuaciones. En tales casos se desiste de obtener una solución analítica del problema, y nos atenemos a la computadora para que nos proporcione soluciones aproximadas con base en técnicas iterativas. Se construye una función de prueba (modelo), se calcula la suma de las diferencias al cuadrado, y luego variamos la función elegida hasta que se encuentre un mínimo para esa suma. La aplicación de estos métodos basados en computadora se utilizaron en el presente proyecto. Con todo, si puede hallarse un método para probar un modelo en forma lineal, la solución será más sencilla.

Adviértase bien que, en todos los casos, es responsabilidad del experimentador escoger el tipo de función que ha de emplearse; todo lo que el método de mínimos cuadrados puede hacer es proporcionar, para la función escogida, los valores de los parámetros que mejor se ajustan a las observaciones.

BUSQUEDA DE FUNCIONES NO LINEALES.

Todo el análisis anterior implicaba la suposición de que ya se poseía un modelo que se deseaba comparar con un sistema. Aunque por lo común éste es el caso, a veces pasa que se tiene un conjunto de observaciones para las cuales no hay un modelo disponible. Esto puede ocurrir, por ejemplo, en la investigación del modelo de un fenómeno que nunca se ha observado antes, o bien si se trabaja en un sistema que es tan complicado que nunca se dispondrá de un modelo teórico. Las observaciones, cuando se grafican en forma elemetal, probablemente muestren una curva que no tenga una forma fácilmente identificable. En ausencia de un modelo, ¿qué se puede hacer?.

Algo que se puede hacer es tratar de encontrar funciones que tengan algún grado de correspondencia con las observaciones. Este procedimiento puede ser muy útil. Por ejemplo. en sistemas muy complejos para los cuales hay poca esperanza de construir modelos teóricos, tal vez sea lo único que podamos hacer. Un modelo empírico, aunque sea sólo una función matemática que no es más que una reformulación del comportamiento real del sistema en forma matemática, puede facilitar el procesamiento de las observaciones por computadora, y será indispensable para procedimientos como la interpelación o extrapolación. Estos modelos se pueden usar, por ejemplo,

para pronosticar la reacción del PNB de un país ante un cambio en la tributación, o para obtener mediciones de temperatura a partir de la curva de calibración de un termómetro de resistencia para instrumentación.

Por otro lado, en sistemas más sencillos, para los cuales existe la esperanza de poder construir un modelo teórico a partir de principios básicos, algunas funciones, cuando se demuestra que son adecuadas a las observaciones, pueden ofrecernos una valiosa guía para construir modelos al sugerir el tipo de procesos físicos que están presentes en el fenómeno. Aún así, debemos ser cautelosos. El hecho de que se haya identificado una función que parece ser consistente con nuestro conjunto de observaciones a un nivel particular de precisión, no "prueba" que se haya encontrado la función "correcta". Muy a menudo, funciones de tipos muy diversos pueden mostrar comportamientos muy parecidos, especialmente en un intervalo corto de las variables, y la "guía" de una función inadecuadamente definida puede ser muy desorientadora. El "asumir" mal esta función, puede retrasar el progreso teórico genuino durante años, y en la historia de la física hay muchos ejemplos de esta falta de comprensión de que cualquier elección de una función empírica debe ser provisional.

Por consiguiente, con la debida atención a la importancia posiblemente limitada de los procedimientos empíricos, se describirá algunos de los métodos que pueden utilizarse. Pueden ser muy sencillos en algunos casos, y dos de ellos son importantes porque tienen que ver con funciones de ocurrencia relativamente común. Supóngase para la discusión siguiente, que se han hecho mediciones de dos variable, que llamaremos x y y.

Funciones de potencia.

Considérese la función:

$$y = x^a$$

donde a es una constante. Se tiene:

$$\log y = a \log x$$

y una grárica de *log y* vs. *log x* es una línea recta de pendiente **a**. En consecuencia, si se quiere comprobar sí una función potencial es una función para las observaciones, se las puede graficar en la forma *log y* vs *log x*.. Si, al graficar en esa forma, los puntos restantes corresponden bien con una línea recta, podemos afirmar que una función potencial es un buen ajuste para nuestras observaciones. El valor del exponente adecuado, **a**, se derivará de la pendiente de la gráfica y se obtendrá dentro de límites de incertidumbre que dependen de la incertidumbre graficada con los puntos. Una gráfica como esta puede trazarse en papel cuadriculado ordinario graficando los valores calculados de *log y* y *log x*, o bien empleando papel para gráficas semi-logarítmicas. Es así como se le aplica el mismo método de mínimos cuadrados.

Funciones exponenciales.

Para muchos fenómenos físicos, lo adecuado es una función exponencial. Considérese:

$$Y = a\, e^{bx}$$

donde a y b son constantes. En este caso

$$\log_e y = \log_e a + bx$$

y la gráfica de la función será una línea recta cuando grafiquemos $\log_e$ y vs. x. Por tanto, si hay razón para sospechar que una función exponencial es adecuada para un sistema particular, debemos hacer una gráfica semilogaritmica, ya sea en papel cuadriculado ordinario, buscando los valores de $\log_e$ y, o en papel para gráficas "semi-log", que tiene una escala logaritmica y una lineal. Los valores adecuados de a y b podrán obtenerse de la ordenada al origen y la pendiente de la recta, con incertidumbres determinadas por las incertidumbres graficadas con los valores medidos.

Representación polinomial.

Si resulta que ni una potencia simple o una función exponencial proporcionan un buen ajuste a un conjunto dado de observaciones, la probabilidad de toparse con una función más compleja que sea la apropiada es muy pequeña. En tales casos, con frecuencia es útil recurrir a una representación polinomial:

$$y = a_0 + a_1 x + a_2 x^2 + ...$$

Aunque una representación como ésta quizás no aporte una visión muy profunda respecto a la base teórica fundamental para la operación del sistema, por lo menos ofrecerá algunas de las ventajas de los modelos empíricos. En todo caso, al menos permite el procesamiento de observaciones por computadora y proporciona una base satisfactoria para la interpolación y la extrapolación.

Los coeficientes de expansión polinomial adecuados para nuestro sistema en particular, pueden determinarse usando el principio de mínimos cuadrados. Ahora bien, debe considerarse que el esfuerzo matemático aumenta de manera directa cuando aparecen muchos términos en el polinomio, pues se debe de hacer un cálculo para cada potencia y luego aplicar un proceso de superposición de la contribución de cada término del polinomio a la forma requerida de la gráfica.

Existe un método alternativo al de mínimos cuadrados cuando la expresión es polinómica. La técnica anterior se conoce como cálculo de las diferencias finitas que se aplica directamente a los puntos observados, y para ajustar el polinomio se usa una tabla de diferencias. Este método se usa cuando no se requiere mucha precisión y la dispersión de las muestras no es muy severa.

ANEXO G: LISTADO DE LAS PARTES PRINCIPALES DEL PROGRAMA.

El programa de esta aplicación está formado por diversos bloques, es decir, que el proyecto se encuentra distribuido en diferentes aplicaciones.

Se crearon diversos archivos ejecutables por medio de Visual C++ y esto permite correr los programas en cualquier PC que tenga Windows 95/ 98, además de no dar una saturación de memoria, pues se habilita solamente el código que se va a usar.

Para facilitar el uso del software diseñado se creo un programa (que se denominó "PADRE") cuyo objetivo es únicamente llamar a las diversas aplicaciones por medio de botones. Para hacer este llamado de programas se recurrió a la familia de funciones _SPAWN, que contiene diversas modalidades de ejecución de aplicaciones EXE , COM y BAT.

Es fundamental señalar que en Visual C++ puede programarse de dos maneras: Usando *programación estructurada clásica* o utilizando *programación orientada a objetos*. Esta última es la que se usa por defecto en Visual C++. Para el caso de este proyecto se hizo una mezcla de ambos estilos debido a que el desarrollo de los programas de procesamiento de imágenes se realizó de manera simultánea con el aprendizaje del lenguaje de programación. Visual C++ permite dos maneras de implementación de programas:

- La primera es "a pie", es decir, digitando todo el código y usando las funciones de implementación de Windows (remítase a investigar sobre las MFCs) que crean ventanas, barras, fondos de pantalla, etc. Esta forma tiene la ventaja de ser muy personalizada (fácil de acomodar a las necesidades personales) pero es muy tediosa en la mayoría de aplicaciones por la gran cantidad de digitación y la facilidad de errores. Los archivos creados de esta forma deben de ser hechos con extensión C y se le debe asociar a un *Workspaces*, por medio de opciones especiales (accesibles con ALT + F7). Para mayores detalles lea un manual de Visual C++ versión 5.0 o superior.
- La segunda forma es creando una aplicación en la que solamente existen objetos (lea sobre el término *Clases* dentro de Turbo C++ o Visual C++). Estos programas pueden ser creados de

forma más sencilla por medio del *App Wizard* que es un asistente que permite crear una aplicación totalmante compatible con Windows (con botones, menús, etc.) de manera rápida. Este asistente crea un esqueleto con código predefinido según el tipo de aplicación que se decidió implementar, que es visualmente funcional pero no en ejecución. Luego se recurre a un segundo asistente denominado *Class Wizard* que permite crear código que realice operaciones con determinado evento (dar un clic, dar dos clic, mover el mouse, etc.). Esta forma es más rápida pero genera incomodidades debido a que los objetos (o si lo prefiere las *Clases*) tienen ciertas reglas de utilización que no dan mucha flexibilidad en ciertas aplicaciones.

La elección de una de las dos formas anteriores depende de la aplicación propia que se desea (no hay una mejor que la otra) y de la experiencia que se tenga con este lenguaje de programación. Para el caso de este proyecto se usaron ambas.

Se desarrollaron los programas usando archivos .H y DLL, es decir, que las funciones de procesamiento de imagen no residen en el programa principal sino que solamente son llamadas por este. Para realizar este "llamado" de funciones es obligatorio hacer un *#include* de las librerías estáticas o archivos .H (en este caso "tesis.h" y "tesis2.h"). Si se tratase de archivos de librería dinámicos o .DLL no se llaman con un *#include* sino que se adicionan al *Workspace* de la aplicación por medio de una serie de opciones de los menús de Visual C++. En ambos casos, el código que forma internamente a las funciones de procesamiento de imágenes se encuentra dentro de esas librerías y digitarlo en páginas es demasiado voluminoso.

Por la razón anterior se han seleccionado algunas porciones del código de aplicación que funcionan por si mismos (se han acomodado para que tengan similitud entre si) y se presentan a continuación. Las porciones que tengan el símbolo " // " son comentarios (este símbolo lo usa Visual C++ para indicar líneas de comentario) que pretenden simplificar la comprensión del programa.

Las líbrerías dinámicas y estáticas no se incluyen por el motivo antes expuesto y los programas funcionarán solamente si se tiene una tarjeta digitalizadora PIXCI-SV4, pues la apertura de la tarjeta desde software verifica una llave de hardware que se tiene en la misma.

PROGRAMA DE SELECCIÓN DE OBJETOS COLOCADOS EN LA COORDENADA "Y" MAS CERCANA AL BRAZO (EJEMPLO DE SELECCIÓN POR POSICION).

```
#define FORMATO    "RS-170"
#define CONFIG          NULL
#include <windows.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <signal.h>
#include <stdlib.h>
#include <stdarg.h>
#include "xcobj.h"
#include "tesis.h"
#include "tesis2.h"
#define Printf  winprintf

//DECLARACION DE FUNCION DE IMPRESIÓN DE CARACTERES EN VENTANA
void winprintf(char *format, ...);

void signaled(int sig)
{
   pxd_close( );
   abort( );
}
//DECLARACION DE VARIABLES
        long int d,n;
        int c,flag, status0,status1,status2,status3,radio,pos0,pos4;
        int  destino[5];
        char    *nombre = "c:\\imagenes\\cali.bmp";
        double ycent, xcent, zcent, increm, menorY;
        int     elegido,cleared = 0;
        int     o,i, h, r, p, p2, value , j, cont, linant, linsig, area ;
        struct pxywindow contornos;
        unsigned long  novale;
        TEXTMETRIC  textmetric;
        HWND    hWnd;
        struct pximage image;
        unsigned long count;
//DEFINICION DE BUFFER DE MEMORIA
        #define A_XDIM          3
        #define A_YDIM          10
        #define COLOR           1
        unsigned  char  colorimage_buf[A_XDIM*A_YDIM*COLOR];

//POSICION DE INICIO DE LOS MOTORES
                int  actual[ ] = {130,255,255,200,127};

//FUNCION PARA MANEJAR EL PUERTO SERIE
        void enviar(unsigned short port, int data)
{
```

```
            int m;
        // 8bits, SIN PARIDAD, 1 stop, Dlab=1
         m=_outp(port+3,0x83);

        // BAUDIOS 9600 (115200/12)
        m=_outp(port,0xc);

        // COLOCA A CERO EL DIVISOR (MS)
        m=_outp(port+1,0);

        //LISTO PARA ENVIAR DATOS
        m=_outp(port+3,3);

        //ENVIAR DATOS
        m=_outp(port,data);
}
// FUNCION PARA VISUALIZAR EN PANTALLA LA IMAGEN
//CREA LA VENTANA DE VIDEO
void do_StretchDIBits(HWND  hWnd)
{
   RECT rect;
   HDC  hDC;
   int  vgabits;
    hDC = GetDC(hWnd);
   GetClientRect(hWnd, &rect);
   rect.right++;
   rect.bottom++;
   SetStretchBltMode(hDC, STRETCH_DELETESCANS);
   pxd_StretchDIBits(1L, 0, 0, -1, -1,
                     0, 0, hDC,
                     rect.right / 4, rect.bottom / 4,  // esquina izquierda
                     rect.right / 2, 3 *rect.bottom / 4,  // tamaño de ventana
                     0);
   vgabits = GetDeviceCaps(hDC, PLANES) * GetDeviceCaps(hDC, BITSPIXEL);
   ReleaseDC(hWnd, hDC);
   if ((pxd_cdim( ) == 1 && vgabits < 16)
    || (pxd_cdim( ) == 3 && vgabits < 24) ) {
        Printf("\n\n");
        Printf("Este modo VGA tiene %d bits por pixel;\n", vgabits);
   }
} //FIN DE FUNCION DE VISUALIZACION EN PANTALLA DE LA IMAGEN.

//PROGRAMA DE LLAMADO A LAS FUNCIONES DE PROCESAMIENTO DE IMAGEN
//Y MOVIMIENTO DE MOTORES.
void func_imagenes(HWND hWnd)
  {
        static  int  k=0;
        static  int  flag;
        if (k==0){
                Printf("Bienvenidos a tesis\n");
                                Printf("sobre RECONOCIMIENTO de OBJETOS\n\n");
                                Printf("Presentado por: Gamero y Alas\n"); }

                signal(SIGFPE, signaled);
        //ACTIVACION  DESDE SOFTWARE DE LA TARJETA DIGITALIZADORA
                i = pxd_xcopen(FORMATO, CONFIG);
```

```
//DETERMINACION DE LA RESOLUCION CON QUE LA TARJETA CAPTURARA LOS DATOS
//SE COLOCA 720 PIXELES HORIZONTALES Y EN LA VERTICAL 480 (2 * 240 DEBIDO
//AL INTERLANCE)
pxd_vidparm( 720, 1,-1, 240, 1, -1, 1);
pxd_video('p',1L);
pxd_video('z', 1L);
pxd_video('p', 0L);
//ABRIR EL BUFFER CON EL FORMATO ESTABLECIDO
   if ((i = pxd_iopen(0, 1L, 30, 40, 30+A_XDIM, 40+A_YDIM, 'r')) != 1) {
         Printf("Se ha producido un error con el formato\n");
   }

//SUSTITUIR PARA IMÁGENES TRASLAPADAS++++++++++++++++++++++

//GUARDAR UNA IMAGEN ANTES DE PROCESAR PARA COLOCAR EN PANTALLA
         i = archivar(nombre, 1L, 0, 0, -1, -1);
//FILTRO DE MEDIANA APLICADA
         i = f_mediana(NULL, defi_imagen(1L, 0, 0, -1, -1), defi_imagen(1L, 0, 0, -1, -1));
         i = f_mediana(NULL, defi_imagen(1L, 0, 0, -1, -1), defi_imagen(1L, 0, 0, -1, -1));
//COPIA DE LA IMAGEN UMBRALIZADA
i = umbral (NULL, defi_imagen(1L, 0,0,-1,-1),defi_imagen(3L, 0,0,-1,-1),0,128,0,255);
//UMBRALIZADO PARA EL PROCESO
//SE USA EL BUFFER 1 PARA EL PROCESAMIENTO, POR ESO SE UMBRALIZA
i = umbral(NULL, defi_imagen(1L, 0,0,-1,-1),defi_imagen(1L, 0,0,-1,-1),0,128,0,255);

//EL RESTO DE CODIGO ES IGUAL PARA LOS OBJETOS TRASLAPADOS
//VER REFERENCIA DE LA PARTE INFERIOR++++++++++++++++++++++

//COLOCAR BUFFER 2 EN BLANCO PARA TRAZAR LAS LINEAS DE CUADRICULA EN EL
i = color_blanco(NULL, defi_imagen(2L, 0,0,-1,-1),0,255,255);

//DEFINE VALORES PARA SER USADOS EN EL RECONOCIMIENTO DE OBJETOS
setupimagen(&image, PXSDIGI, 1L);
sxy.x = -1;
sxy.y = 0;
contornos.nw.x = 3;
contornos.nw.y = 3;
contornos.se.x = 100;
contornos.se.y = 100;
//FUNCION QUE DETERMINA EL AREA, CENTROIDE Y NUMERO DE OBJETOS
         r = objetos (NULL, &image, &sxy, 'e'^'q',0,0, NULL, 255, NULL, 20, results, &novale);
//TERMINADA LA FUNCION SE TIENE EN LA VARIABLE results  LOS VALORES CALCULADOS
//PARA CADA UNO DE LOS OBJETOS


//LAZO QUE PERMITE ELEGIR  LA PRIMERA VEZ EL OBJETO QUE SE  SELECCIONA EN //BASE AL
CRITERO Y LA SEGUNDA VEZ SE ESCOGE LA POSICION EN QUE SE //ENCUENTRA LA MARCA
PARA DEPOSITAR EL OBJETO.
for ( o = 1; o <= 2; o++) {

//AQUÍ SI SE CAMBIA LA CONDICION, LA SELECCIÓN SE HACE EN BASE A OTRO
//CRITERIO
```

```
//ESTA MARCA SE USA COMO REFERENCIA POSTERIOR>>>>>>>>>>>>>>>>>>>
//SE UTILIZA EN LAS EXPLICACIONES SIGUIENTES

if ( o = =  1) {
//SE REALIZA UNA SELECCIÓN DEL OBJETO QUE SE ENCUENTRE MAS CERCA DEL BRAZO
//AL VALOR DE LA MARCA DE COLOCACION DE OBJETOS
//LA SELECCIÓN EN ESTE CASO ES POR POSICION
                menorY=9999;
//VERIFICAR EN TODOS LOS VALORES QUE SE BUSCARON
        for (j=0; j<r; j++){
//EL TAMAÑO DE LA MARCA ES 1200, SE LE DISCRIMINA DE LA SELECCIÓN POR MEDIO
//DE ESTE IF
        if(results[j].xyarea>=1250){
//LA FILA DONDE TERMINA EL ENTORNO ES 103 (COORDENADA y DE PIXELES)
                if(((int) results[ j ].ucom.yd<=menorY)&&(results[ j ].ucom.yd>=103)){
                        menorY = ( int ) results[ i ].ucom.yd;
//CAPTURAR EL INDICE QUE TIENE A ESE OBJETO MENOR
                        i = j; } }
        }
} //FIN DE LA SELECCIÓN DE OBJETO

//BUSQUEDA DE LA MARCA A DONDE SE DEPOSITARA EL OBJETO
if (o = = 2) {
                area=9999;
//VERIFICAR EN TODOS LOS VALORES QUE SE BUSCARON
        for (j=0; j<r; j++){
//SE SABE QUE LA MARCA TIENE UN TAMAñO MENOR A 1200 PIXELES
//POR LO TANTO SE LE BUSCA POR TAMAñO
        if( (results[j].xyarea< 1250)&&( results[j].xyarea>=100) ) {
                if ((int) results[ j ].xyarea< =area){
                        area = results[ j ].xyarea;
//CAPTURAR EL INDICE DE LA MARCA
                        i = j; } }
        }
} //FIN DE LA SELECCIÓN DE LA MARCA

        elegido = i;

//TRAZAR EN EL BUFFER 2 LAS LINEAS
                        for (h=0; h<=45; h++){
                                for (j=0; j<=719; j++){
                                        poner_pix(2L, j,( int )(0.08115*h*h+4.42833*h+107),0); } }
//CUENTA DE DIMENSION EN Y.
        cont = 0;
        if(results[ i ].xyarea!=0){
        for(h=107; h<=479; h++){
                        p=sacar_pix ( 2L,( int ) results[ i ].ucom.xd, h);
                        if(p==0){
                                if(h<=( int )results[i].wind.se.y){
                                        cont++;}
                                if(h>( int ) results[i].wind.se.y){
                                linsig=( int )(0.08115*cont*cont+4.42833*cont+107);
                                linant=( int )(0.08115*(cont-1)*(cont-1)+4.42833*(cont-1)+107);
        ycent = (double)cont+((double)(results[i].wind.se.y-linant)) / ((double)(linsig-linant));
                break; }
                        }
```

```
        } }
        //SE EJECUTA SI NO HAY OBJETOS
        if(results[ i ].xyarea = =0) {
                ycent=0;  }
//FIN DE OBTENER DIMENSION EN Y

        //COORDENADA Z
//SE CALCULA EL VALOR DE ESCALA PARA LA DIMENSION Z
increm = -0.0000512775*ycent*ycent*ycent+0.00390345*ycent*ycent-0.0001288508*ycent+10.9856;
        zcent= (double)(results[ i ].wind.se.y - results[ i ].ucom.yd)/increm;
//FIN DE DIMENSION EN Z

        //COORDENADA X
value = 0;
i = color_blanco(NULL, defi_imagen(1L, 0,0,-1,-1),defi_imagen(4L, 0,0,-1,-1),0,255,255);
        setupimagen(&image, PXSDIGI, 4L);
        for (h=0; h<=21; h++){
        sxy.x = 360+14*h+ (int)0.44*h;
        sxy.y = 103;
        exy.x = 360+22*h+ (int)0.31*h;
        exy.y = 479;
//TRAZA LAS LINEAS DE LA COORDENADA X
        i = lineasx(NULL,&image, &sxy, &exy, 1, 3, 's', &value, NULL, NULL); }
                        for (h=1; h<=21; h++){
        sxy.x = 360-14*h- (int)0.44*h;
        sxy.y = 103;
        exy.x = 360-22*h- (int)0.31*h;
        exy.y = 479;
        i = lineasx(NULL,&image, &sxy, &exy, 1, 3, 's', &value, NULL, NULL); }
        i=elegido;
        cont = 0;
        if(results[ i ].ucom.xd>=360) {
        linant = 360;
                for(h=361; h<=718; h++){
                p=sacar_pix(4L,h,(int) results[ i ].ucom.yd);
                p2=sacar_pix(4L,h+1,(int) results[ i ].ucom.yd);
                if((p2-p)<0){
                        cont ++;
                        linsig=h+1;
                        if((results[ i ].ucom.xd<(h+4))&&(results[ i ].ucom.xd>h)) {
                                xcent = cont;
                                break;}
                        if(h>results[ i ].ucom.xd) {
        xcent = (double)(cont-1)- 0.5 + (double)(results[ i ].ucom.xd-linant)/(double)(linsig-linant);
                                break;}
                }
                linant = linsig;
                }
                if((results[i].ucom.xd>=360)&&(results[ i ].ucom.xd<=363))xcent =0.0;
        }
        if(results[ i ].ucom.xd<360){
        linant = 360;
                for(h=359; h>=1; h--) {
                p=sacar_pix(4L,h,(int) results[i].ucom.yd);
                p2=sacar_pix(4L,h-1,(int) results[i].ucom.yd);
                if((p2-p)<0){
```

```
                    cont ++;
                    linsig=h-1;
                    if((results[i].ucom.xd>(h-4))&&(results[i].ucom.xd<h)){
        xcent = (-1.0)*cont;
                            break;}
                    if(h<results[i].ucom.xd){
        xcent = (double)(cont-1)  + (double)(results[i].ucom.xd-linant)/(double)(linsig-linant);
        xcent = xcent * (-1.0);
                            break;}
                }
                linant = linsig;
                }
        if((results[i].ucom.xd>=357)&&(results[i].ucom.xd<=359))xcent =0.0;
        }

// FIN DE COORDENADA X

//CALCULO DEL RADIO Y DEL ANGULO A PARTIR DE LA X y Y
        radio = (int)(sqrt(xcent*xcent+ycent*ycent));
//LIMITA LOS VALORES DEL RADIO A LAS DIMENSIONES DE ENTORNO
if(radio>38) radio = 38;
if(radio<19) radio = 19;
//POSICION DEL MOTOR DE LA BASE (MOTOR  #0)
pos0 = 130-(int)((180*(atan(xcent/(ycent-12))) / 3.14159) / 0.418);
if(pos0>=170) pos0 =pos0+6;
if(pos0<=31) pos0 =pos0-6;
        if (pos0>255) pos0 = 255;
if (pos0<0) pos0 = 0;



//VALORES PARA LOS MOTORES #1, #2 y #3
//ASIGNA A CADA MOTOR UN ESTADO DE APAGADO
status0=0;
status1=0;
status2=0;
status3=0;

destino[0]=pos0;
//VALORES PARA DE DESTINO PARA LOS MOTORES
//DESTINO DE MOTOR #1 (ECUACION UNICA)
destino[1]= (int)(-0.17072177*radio*radio - 1.4039895*radio + 312.44215);
//DESTINO DE MOTOR #2  (ECUACIONES SECCIONADAS)
if((radio>=19)&&(radio<=34)){
        destino[2]= (int)(-0.0038203*radio*radio*radio*radio + 0.371616*radio*radio*radio
13.828639*radio*radio +228.327181*radio - 1141.784353);}
if((radio>34)&&(radio<=38)){
        destino[2]= (int)(-2.75*radio*radio*radio + 296.214285*radio*radio - 10645.178571*radio +
127732.685714);}
//DESTINO DE MOTOR #3 (ECUACIONES SECCIONADAS)
if((radio>=19)&&(radio<31)){
        destino[3]=(int)(-0.021992618*radio*radio*radio + 1.2913284*radio*radio - 12.38749824*radio -
67.7121228);}
if((radio>=31)&&(radio<=35)){
        destino[3]= (int)(0.41666667*radio*radio*radio - 45.178571*radio*radio + 1612.619047*radio -
18847.857143);}
```

```
if((radio>35)&&(radio<=38)){
        destino[3]= (int)(-2.5136612*radio*radio*radio + 266.948297*radio*radio - 9431.06137*radio +
110964.06998);}
if(radio>23) destino[3] = destino[3]-25;


//ABRIR LA PINZA DEL ROBOT
actual[4]=90;
enviar(0x3f8,255);
enviar(0x3f8,4);
enviar(0x3f8,actual[4]);
//ASIGNAR A LOS MOTORES QUE SE DEBEN MOVER EL ESTADO DE 1
        if ((destino[0]-actual[0])!=0){status0=1;}
        if ((destino[1]-actual[1])!=0){status1=1;}
        if ((destino[2]-actual[2])!=0){status2=1;}
        if ((destino[3]-actual[3])!=0){status3=1;}


        flag = status0+status1+status2;
        if(ycent<0){
        flag = 0;
        status3=0;}
//EL LAZO SE REALIZA SI AL MENOS UN MOTOR SE DEBE MOVER.
        while (flag>0){

// MOTOR # 0
        if ((destino[0]-actual[0])>0){
                actual[0]=actual[0]+1;
enviar(0x3f8,255);
enviar(0x3f8,0);
enviar(0x3f8,actual[0]);}
                if ((destino[0]-actual[0])<0){
                        actual[0]=actual[0]-1;
enviar(0x3f8,255);
enviar(0x3f8,0);
enviar(0x3f8,actual[0]);}

for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[0]-actual[0])==0){status0=0;}

// MOTOR #1
        if ((destino[1]-actual[1])>0){
                actual[1]=actual[1]+1;
enviar(0x3f8,255);
enviar(0x3f8,1);
enviar(0x3f8,actual[1]);}
                if ((destino[1]-actual[1])<0){
                        actual[1]- -;
enviar(0x3f8,255);
enviar(0x3f8,1);
enviar(0x3f8,actual[1]);}
for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}
```

```
        if ((destino[1]-actual[1])==0){status1=0;}

// MOTOR # 2
        if ((destino[2]-actual[2])>0){
                actual[2]++;
enviar(0x3f8,255);
enviar(0x3f8,2);
enviar(0x3f8,actual[2]);}
                if ((destino[2]-actual[2])<0){
                        actual[2] - -;
enviar(0x3f8,255);
enviar(0x3f8,2);
enviar(0x3f8,actual[2]);}

                for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[2]-actual[2])==0){status2=0;}


        flag = status0+status1+status2;

}
//MOVER EL MOTOR QUE CONTROLA  LA MUñECA
        while(status3>0){
        // motor 3
        if ((destino[3]-actual[3])>0){
                actual[3]++;
enviar(0x3f8,255);
enviar(0x3f8,3);
enviar(0x3f8,actual[3]);}
                if ((destino[3]-actual[3])<0){
        actual[3]--;
enviar(0x3f8,255);
enviar(0x3f8,3);
enviar(0x3f8,actual[3]);}

for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[3]-actual[3])==0){status3=0;}
        }
        //CERRAR LA PINZA
        actual[4]=0;
enviar(0x3f8,255);
enviar(0x3f8,4);
enviar(0x3f8,actual[4]);

//INICIA EL PROCESO DE REGRESAR EL BRAZO A SU POSICION DE REPOSO
//CON EL OBJETO AGARRADO
destino[1]=255;
destino[2]=255;
destino[3]=200;


        if ((destino[0]-actual[0])!=0){status0=1;}
```

```
        if ((destino[1]-actual[1])!=0){status1=1;}
        if ((destino[2]-actual[2])!=0){status2=1;}
        if ((destino[3]-actual[3])!=0){status3=1;}

        flag = status0+status1+status2+status3;
                if(ycent<0){
        flag = 0;
        status3=0;}
//INICIA EL MOVIMIENTO DE REGRESO DEL BRAZO
        while (flag>0){

// MOTOR # 0
        if ((destino[0]-actual[0])>0){
                actual[0]=actual[0]+1;
enviar(0x3f8,255);
enviar(0x3f8,0);
enviar(0x3f8,actual[0]);}
                if ((destino[0]-actual[0])<0){
                        actual[0]=actual[0]-1;
enviar(0x3f8,255);
enviar(0x3f8,0);
enviar(0x3f8,actual[0]);}

for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[0]-actual[0])==0){status0=0;}

// MOTOR #1
        if ((destino[1]-actual[1])>0){
                actual[1]=actual[1]+1;
enviar(0x3f8,255);
enviar(0x3f8,1);
enviar(0x3f8,actual[1]);}
                if ((destino[1]-actual[1])<0){
                        actual[1]--;
enviar(0x3f8,255);
enviar(0x3f8,1);
enviar(0x3f8,actual[1]);}
for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[1]-actual[1])==0){status1=0;}

// MOTOR #2
        if ((destino[2]-actual[2])>0){
                actual[2]++;
enviar(0x3f8,255);
enviar(0x3f8,2);
enviar(0x3f8,actual[2]);}
                if ((destino[2]-actual[2])<0){
                        actual[2]--;
enviar(0x3f8,255);
enviar(0x3f8,2);
enviar(0x3f8,actual[2]);}
```

```
                for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[2]-actual[2])==0){status2=0;}

// MOTOR #3
        if ((destino[3]-actual[3])>0){
                actual[3]++;
enviar(0x3f8,255);
enviar(0x3f8,3);
enviar(0x3f8,actual[3]);}
                if ((destino[3]-actual[3])<0){
                        actual[3]--;
enviar(0x3f8,255);
enviar(0x3f8,3);
enviar(0x3f8,actual[3]);}

for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[3]-actual[3])==0){status3=0;}

        flag = status0+status1+status2+status3;

}

} //SE CIERRA EL FOR QUE MANEJA EL DOBLE PROCESO DE VERIFIACAR LA POSICION DE //LA
MARCA Y  SELECCIONAR AL OBJETO EN BASE A UN CRITERIO.
// EN AMBOS CASOS SE MUEVE EL BRAZO, PARA TOMAR EL OBJETO Y PARA //DEPOSITARLO
EN LA MARCA.
                k++;
} //TERMINA LA FUNCION DE LLAMADO DEL PROCESAMIENTO

//FUNCION PARA PRESENTAR CARACTERES EN PANTALLA
//YA SEA NUMEROS O LETRAS.
  void winprintf(char *format, ...)
  {
        HDC     hDC;
        SIZE    size;
        va_list argp;
        char    buf[200];
        char    *p;
        int     beeps = 0;
        int     nl = 0;

        va_start(argp, format);
        if ((vsprintf(buf, format, argp)+1) > sizeof(buf)) {
           va_end(argp);
           FatalAppExit(0, "Se excedio la capacidad");
        }
        va_end(argp);

        hDC = GetDC(hWnd);
        SelectObject(hDC, GetStockObject(OEM_FIXED_FONT));
        if (!cleared) {
           RECT    rect;
```

```
            GetTextMetrics(hDC, &textmetric);
            SetRect(&rect, 0, 0, 32767, 32767);
            FillRect(hDC, &rect, GetStockObject(WHITE_BRUSH));
            ydrawpos = 1;
            xdrawpos = GetDeviceCaps(hDC, LOGPIXELSX) / 8;
            cleared++;
        }
        for (p = buf; *p; p++) {
            if (*p == '\a')
                *p = ' ', beeps++;
            if (*p == '\n')
                *p = ' ', nl++;
        }

        TextOut(hDC, xdrawpos, ydrawpos, buf, strlen(buf));
        GetTextExtentPoint32(hDC, buf, strlen(buf), &size);
        xdrawpos += size.cx;

        while (beeps--)
            MessageBeep((uint)-1);
        while (nl--) {
            ydrawpos += textmetric.tmExternalLeading + textmetric.tmHeight;
            xdrawpos = GetDeviceCaps(hDC, LOGPIXELSX) / 8;
        }
        ReleaseDC(hWnd, hDC);
    }


//ESTABLECE LAS PROPIEDADES DE LA VENTANA DE WINDOWS QUE SE CREARA
        LRESULT CALLBACK MainWndProc(
        HWND    hWnd,
        unsigned   message,
        WPARAM          wParam,
        LPARAM          lParam)  {
        PAINTSTRUCT paints;
        RECT   rects;
        HDC    hDC;

        switch (message) {
//PERMITE LLAMAR EN FORMA SUCESIVA  LA APLICACIÓN CON CLICK DEL MOUSE
//O POR MEDIO DEL TECLADO
            case WM_CHAR:
            case WM_LBUTTONDOWN:
            case WM_RBUTTONDOWN:
                func_imagenes(hWnd);
                break;

            case WM_DESTROY:
                PostQuitMessage(0);
                break;

            case WM_CREATE:
                break;
            case WM_PAINT:
                hDC = BeginPaint(hWnd, &paints);
                GetWindowRect(hWnd, &rects);
```

```
                DrawText(hDC, " Presione una tecla o haga clic", -1, &rects, DT_LEFT|DT_VCENTER);
                EndPaint(hWnd, &paints);
                break;

            default:
                return(DefWindowProc(hWnd, message, wParam, lParam));
        }
        return(0);
    }

//CREA LA VENTANA DE APLICACIÓN POR MEDIO DE LAS LIBRERIAS CON LAS
//CUALES HA SIDO CREADO WINDOWS
    int PASCAL WinMain(
        HANDLE      hInstance,
        HANDLE      hPrevInstance,
        LPSTR lpCmdLine,
        int     nCmdShow    ){
        MSG   msg;
        WNDCLASS  wc;

        if (hPrevInstance)
            return(FALSE);

        wc.style = CS_BYTEALIGNWINDOW;
        wc.lpfnWndProc = MainWndProc;
        wc.cbClsExtra = 0;
        wc.cbWndExtra = 0;
        wc.hInstance = hInstance;
        wc.hIcon = 0;
        wc.hCursor = LoadCursor(0, IDC_ARROW);
        wc.hbrBackground = GetStockObject(WHITE_BRUSH);
        wc.lpszMenuName =  NULL;
        wc.lpszClassName = "PXobjWClass";

        if (!RegisterClass(&wc))
            return (FALSE);

        hWnd = CreateWindow("PXobjWClass",
                            "Universidad Don Bosco. Electronica",
                            WS_OVERLAPPEDWINDOW,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            0, 0, hInstance, NULL );

        if (!hWnd)
            return (FALSE);
//MUESTRA Y ACTUALIZA LA VENTANA ESTILO WINDOWS DE LA APLICACION
        ShowWindow(hWnd, nCmdShow);
        UpdateWindow(hWnd);
//EJECUCION DEL PROGRAMA DE PROCESAMIENTO DE LAS IMAGENES
        func_imagenes(hWnd);

        while (GetMessage(&msg, 0, 0, 0)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
```

```
        return (msg.wParam);
}
```

Para hacer otros tipos de selección, como por ejemplo: tomar el objeto más grande, el que se encuentra más a la derecha, más cerca de la cámara, etc.; bastará con cambiar la parte de selección de la figura y todo el resto del proceso se realizará de idéntica forma.

Para el caso se muestra una selección por tamaño, en la cual el brazo tomará el objeto más grande que encuentre en la escena.

```
//SUSTITUIR DESDE DONDE APARACE >>>>>>>>>>>>>>>>>>>
//EN EL PROGRAMA ANTES LISTADO
if ( o = =  1) {
//SE REALIZA UNA SELECCIÓN DEL OBJETO MAS GRANDE QUE SE ENCUENTRE EN LA
//ESCENA y ESTE SERA EL OBJETO A TOMAR.
                mayorA=9999;
//VERIFICAR EN TODOS LOS VALORES QUE SE ENCONTRARON DE OBJETOS
        for (j=0; j<r; j++){
//EL TAMAÑO DE LA MARCA ES 1200, SE LE DISCRIMINA DE LA SELECCIÓN POR MEDIO
//DE ESTE IF
        if(results[j].xyarea>=1250){
                if (results[ j ].xyarea > mayorA) {
                        mayorA =  results[ i ].xyarea;
//CAPTURAR EL INDICE QUE TIENE A ESE OBJETO MAS GRANDE
                        i = j; } }
        }
} //FIN DE LA SELECCIÓN DE OBJETO
```

- Nota1 : Es necesario que se defina la variable mayorA como dato tipo *int.*
- Nota2 : Las otras posibles condiciones de selección no se colocan por ahorro de espacio y poque son muy semejantes a ésta en lógica.

El programa anterior no puede distinguir entre dos objetos traslapados, la gran mayoría de ocasiones lo presentará como uno sólo.

Al listado previo se le puede hacer una serie de modifcaciones en la porción de captura y procesamiento de la imagen para permitirle reconocer objetos traslapados. Básicamente la modificación es:

- Quitar los filtros de la mediana y dejar la imagen como se encuentra. Esto se hace para evitar que los contornos se suavicen y se tienda a uniformizar a los objetos traslapados.
- Sacar dos copias de la misma imagen para realizar dos procesos separados.
- Uno de los buffer se umbraliza como antes y el otro se le procesa por medio del operador de Kirsch para resaltar sus contornos y detalles.
- Ambas imágenes se encuentran umbralizadas: la primera tiene el fondo blanco y las figuras son negras; la segunda tiene en blanco todos los cambios (contornos, etc.) y las partes homogéneas en negro.
- Luego se realiza un proceso de separación de los objetos aplicando operaciones booleanas a ambas imágenes y guardando el resultado para realizar el procesamiento con él.
- Debido a que quedan pequeñas uniones entre las figuras se realiza un proceso de erosión para separarlos. Este proceso modifica la forma real de las figuras.

El programa se ha hecho por separado debido a que presenta una gran sensibilidad al ruido y a variaciones mínimas en la iluminación, debido a que los cambios son resaltados. También por el proceso de erosión que se necesita en este programa se producen deformaciones notorias de los objetos.

La porción que cambia del listado anterior con el presente programa se muestra a continuación, remítase a buscar la referencia indicada en el programa previo para saber el lugar donde debe colocar este código.

PORCION DE PROGRAMA QUE PERMITE DIFERENCIAR OBJETOS TRASLAPADOS.

```
//SUSTITUIR ESTE CODIGO DONDE SE MUESTREN++++++++++++++++++++
//SE INICIA EL PROCESO HACIENDO UNA COPIA ADICIONAL DE LA IMAGEN TOMADA
//PARA PRESENTARLA EN PANTALLA.
        i = copiar_buffer(NULL, defi_imagen(1L, 0,0,-1,-1),defi_imagen(3L, 0,0,-1,-1));
        i = archivar(name, 1L, 0, 0, -1, -1);

                        //INICIO DE SEPARACION DE OBJETOS
//APLICA EL OPERADOR DE KIRSCH A UNA DE LAS COPIAS DE LA IMAGEN TOMADA.
//EL RESULTADO SE COLOCA EN EL BUFFER 5
i =oper_Kirsch(NULL, defi_imagen(3L, 0,0,-1,-1),defi_imagen(5L, 0,0,-1,-1),0,128,0,255);

//SE  UMBRALIZA LA IMAGEN DE LA OTRA COPIA DE LA IMAGEN
i = umbral(NULL, defi_imagen(1L, 0,0,-1,-1),defi_imagen(1L, 0,0,-1,-1),0,128,0,255);
//SE UMBRALIZA LA IMAGEN RESULTANTE DEL KIRSCH Y SE GUARDA  EN BUFFER 5
i = umbral(NULL, defi_imagen(5L, 0,0,-1,-1),defi_imagen(5L, 0,0,-1,-1),100,200,0,255);
//SE REALIZAN OPERACIONES BOOLENAS PARA SEPARAR LAS FIGURAS Y SE ESCRIBE
//EL RESULTADO SOBRE EL BUFFER 1
i = oper_separar(NULL,defi_imagen(5L, 0,0,-1,-1), defi_imagen(1L, 0,0,-1,-1),defi_imagen(1L, 0,0,-1,-1));
//REALIZAR PROCESO DE EROSION  SUCESIVA A LA IMAGEN PARA SEPARAR DE MANERA
//DEFINITIVA A LOS OBJETOS.
i = erosion(NULL, defi_imagen(5L, 0, 0, -1, -1), defi_imagen(5L, 0, 0, -1, -1));
i = erosion(NULL, defi_imagen(5L, 0, 0, -1, -1), defi_imagen(5L, 0, 0, -1, -1));
i = erosion(NULL, defi_imagen(5L, 0, 0, -1, -1), defi_imagen(5L, 0, 0, -1, -1));
i = erosion(NULL, defi_imagen(5L, 0, 0, -1, -1), defi_imagen(5L, 0, 0, -1, -1));

//AQUÍ CONTINUA EL RESTO DEL PROGRAMA ANTERIOR
```

A continuación se coloca el listado del programa que permite seguir los objetos. Este programa tiene como objetivo ejemplificar el reconocimiento de objetos en movimiento pero solamente puede realizarlo con uno.

Esta aplicación tiene semejanza con la anterior pero difiere en aquellos puntos del programa en donde se requiere hacer repeticiones (por el procesamiento continuo) y porque el programa termina hasta que el objeto sale de la escena.

Esta aplicación no presenta nada en pantalla (ni video, ni ventana) para permitir que el procesamiento continuo de imágenes sea más ágil.

LISTADO DEL PROGRAMA PARA EL SEGUIMIENTO DE UN OBJETO EN MOVIMIENTO.

```
#define FORMATO      "RS-170"
#define CONFIG       NULL
#include <windows.h>
#include <conio.h>
#include <stdio.h>
#include <process.h>
#include <math.h>
#include <signal.h>
#include <stdlib.h>
#include <stdarg.h>
#include "xcobj.h"
#include "tesis2.h"
#include "tesis.h"


void signaled(int sig)
{
   pxd_close( );
   abort( );
}

        long int d,n;
        int c,flag, status0,status1,status2,status3,radio,pos0,pos4;
        int  destino[5];
        double ycent, xcent, zcent, increm, mayorarea;
        unsigned long valores, valores2;
        uint value = 200;
        int    ydrawpos = 1;
        int    xdrawpos = 0;
        int    elegido,cleared = 0;
        int    i, h, r,p,p2,j,cont,linant,linsig;
        struct pximage image;
        struct pxy sxy, exy;
        struct pxywindow contornos;
        unsigned long ngood, novale;
        TEXTMETRIC  textmetric;
        HWND   hWnd;
        struct pximage image;
        unsigned long count;

        #define AOI_XDIM     3
        #define AOI_YDIM     10
        #define COLOR            1
        uchar   colorimage_buf[A_XDIM*A_YDIM*COLOR];
//FUNCION PARA ENVIO DE DATOS AL PUERTO SERIE
        void enviar(unsigned short port, int data)
        {
                int m;
        // 8bits, NO PARIDAD, 1 stop, Dlab=1
         m=_outp(port+3,0x83);
```

```
        // BAUDIOS 9600 (115200/12)
        m=_outp(port,0xc);

        // COLOCAR A CERO DIVISOR (MS)
        m=_outp(port+1,0);

        //LISTO PARA ENVIAR DATO POR EL PUERTO
        m=_outp(port+3,3);

        //ENVIAR DATO
        m=_outp(port,data);
}

//FUNCION QUE REALIZA EL PROCESAMIENTO DE IMAGENES

void prox_func(HWND hWnd)
  {
        static int k=0;
        static int  flag;

          signal(SIGFPE, signaled);
                pxd_vidfield(0);
//ABRIR LA TARJETA DE VIDEO
                i = pxd_xcopen(FORMATO, CONFIG);
                pxd_imsize();

//COLOCAR LA RESOLUCION
        pxd_vidparm(720,1,-1,240,1,-1,1);
//PARA QUE EL BRAZO PUEDA SEGUIR AL OBJETO SE REALIZA UN LAZO CONDICIONAL
//D0 – WHILE QUE FINALIZARA HASTA QUE EL OBJETO SALGA DE LA ESCENA

do{
//CAPTURA EL VIDEO, CADA VEZ QUE SE EJECUTE
        pxd_video('p',1L);
        pxd_video('z', 1L);
        pxd_video('p', 0L);

i = pxd_iopen(0, 1L, 30, 40, 30+A_XDIM, 40+A_YDIM, 'r');
i = f_mediana(NULL, defi_imagen(1L, 0, 0, -1, -1),defi_imagen(1L, 0, 0, -1, -1));
i = f_mediana(NULL, defi_imagen(1L, 0, 0, -1, -1),defi_imagen(1L, 0, 0, -1, -1));
i = f_mediana(NULL, defi_imagen(1L, 0, 0, -1, -1),defi_imagen(1L, 0, 0, -1, -1));

i = umbral(NULL, defi_imagen(1L, 0,0,-1,1),defi_imagen(3L, 0,0,-1,-1),0,128,0,255);
i = umbral(NULL, defi_imagen(1L, 0,0,-1,1),defi_imagen(1L, 0,0,-1,-1),0,128,0,255);
i = color_blanco(NULL, defi_imagen(1L, 0,0,-1,-1),defi_imagen(2L, 0,0,-1,-1),0,255,255);

        setupimagen(&image, PXSDIGI, 1L);
sxy.x = -1;
sxy.y = 0;
contornos.nw.x = 3;
contornos.nw.y = 3;
contornos.se.x = 100;
contornos.se.y = 100;
ngood = 0;
r = objetos(NULL, &image, &sxy, 'e'^'q',0,0, NULL, 255, NULL, 20, results, &novale);
```

```
//DEBIDO A QUE SE TIENE UN SOLO OBJETO SE COLOCA DE UNA SOLA VEZ
//COMO OBJETO SELECCIONADO EL VALOR DEL PRIMERO Y UNICO CALCULADO
i=0;
elegido = i;
//TRAZADO DE LAS LINEAS EN Y
        for (h=0; h<=45; h++){
                for (j=0; j<=719; j++) {
                poner_pix(2L,j,(int)(0.08115*h*h+4.42833*h+107),0); } }

        //CUENTA DE DIMENSION EN Y.
        cont = 0;
        if(results[i].xyarea!=0){
        for(h=107; h<=479; h++) {
                        p=sacar_pix(2L,(int) results[i].ucom.xd, h);
                        if(p==0) {
                                if(h<=(int)results[i].wind.se.y) {
                                        cont++; }
                                if(h>(int)results[i].wind.se.y) {
                linsig=(int)(0.08115*cont*cont+4.42833*cont+107);
                linant=(int)(0.08115*(cont-1)*(cont-1)+4.42833*(cont-1)+107);
                ycent = (double)cont+((double)(results[i].wind.se.y-linant))/((double)(linsig-linant));
                break;  }
                        }
        } }
        //SI NO HUBIERA OBJETOS
        if(results[i].xyarea==0){
                ycent=0;}
//FIN DE DIMENSION EN Y

        //COORDENADA Z
        increm = -0.0000512775*ycent*ycent*ycent+0.00390345*ycent*ycent-
0.0001288508*ycent+10.9856;
        zcent= (double)(results[i].wind.se.y - results[i].ucom.yd)/increm;
//FIN DE DIMENSION EN Z

        //COORDENADA X
i = color_blanco(NULL, defi_imagen(1L, 0,0,-1,-1),defi_imagen(2L, 0,0,-1,-1),0,255,255);
        setupimagen(&image, PXSDIGI, 2L);

        for (h=0; h<=21; h++){
        sxy.x = 360+14*h+ (int)0.44*h;
        sxy.y = 103;
        exy.x = 360+22*h+ (int)0.31*h;
        exy.y = 479;
        i = lineas(NULL,&image, &sxy, &exy, 1, 3, 's', &value, NULL, NULL); }
        for (h=1; h<=21; h++){
        sxy.x = 360-14*h- (int)0.44*h;
        sxy.y = 103;
        exy.x = 360-22*h- (int)0.31*h;
        exy.y = 479;
        i = lineas(NULL,&image, &sxy, &exy, 1, 3, 's', &value, NULL, NULL); }
        i=elegido;
        cont = 0;
        if(results[i].ucom.xd>=360){
        linant = 360;
```

```
		for(h=361; h<=718; h++){
		p=sacar_pix(2L,h,(int) results[i].ucom.yd);
		p2=sacar_pix(2L,h+1,(int) results[i].ucom.yd);
		if((p2-p)<0){
			cont ++;
			linsig=h+1;
			if((results[i].ucom.xd<(h+4))&&(results[i].ucom.xd>h)){
				xcent = cont;
				break;}
			if(h>results[i].ucom.xd){
	xcent = (double)(cont-1)- 0.5 + (double)(results[i].ucom.xd-linant)/(double)(linsig-linant);
				break;}
		}
		linant = linsig;
		}
		if((results[i].ucom.xd>=360)&&(results[i].ucom.xd<=363))xcent =0.0;
	}
	if(results[i].ucom.xd<360){
	linant = 360;
		for(h=359; h>=1; h--){
		p=sacar_pix(2L,h,(int) results[i].ucom.yd);
		p2=sacar_pix(2L,h-1,(int) results[i].ucom.yd);
		if((p2-p)<0){
			cont ++;
			linsig=h-1;
			if((results[i].ucom.xd>(h-4))&&(results[i].ucom.xd<h)){
	xcent = (-1.0)*cont;
				break;}
			if(h<results[i].ucom.xd){
	xcent = (double)(cont-1)  + (double)(results[i].ucom.xd-linant)/(double)(linsig-linant);
	xcent = xcent * (-1.0);
				break;}
		}
		linant = linsig;
		}
	if((results[i].ucom.xd>=357)&&(results[i].ucom.xd<=359))xcent =0.0;
	}


//CALCULO DEL RADIO Y DEL ANGULO
	radio = (int)(sqrt(xcent*xcent+ycent*ycent));
if(radio>38) radio = 38;
if(radio<19) radio = 19;
pos0 = 130-(int)((180* (atan(xcent/(ycent-12))) / 3.14159) /  0.418);

if(pos0>=170) pos0 =pos0+6;
if(pos0<=31) pos0 =pos0-6;

	if (pos0>255) pos0 = 255;
if (pos0<0) pos0 = 0;
//VALORES PARA LOS MOTORES
//ASIGNAR UN ESTADO DE REPOSO POR DEFECTO A TODOS LOS MOTORES
status0=0;
status1=0;
status2=0;
status3=0;
```

```
//ASIGNACION DE LAS POSICIONES DE DESTINO A CADA UNO DE LOS MOTORES
destino[0]=pos0;

destino[1]= (int)(-0.17072177*radio*radio - 1.4039895*radio + 312.44215);

if((radio>=19)&&(radio<=34)){
destino[2]= (int)(-0.0038203*radio*radio*radio*radio + 0.371616*radio*radio*radio -
13.828639*radio*radio +228.327181*radio - 1141.784353);}
if((radio>34)&&(radio<=38)){
destino[2]= (int)(-2.75*radio*radio*radio + 296.214285*radio*radio - 10645.178571*radio +
127732.685714);}

if((radio>=19)&&(radio<31)){
destino[3]=(int)(-0.021992618*radio*radio*radio + 1.2913284*radio*radio - 12.38749824*radio -
67.7121228);}
if((radio>=31)&&(radio<=35)){
destino[3]= (int)(0.41666667*radio*radio*radio - 45.178571*radio*radio + 1612.619047*radio -
18847.857143);}
if((radio>35)&&(radio<=38)){
destino[3]= (int)(-2.5136612*radio*radio*radio + 266.948297*radio*radio - 9431.06137*radio +
110964.06998);}
if(radio>23) destino[3] = destino[3]-25;


//EJECUTAR EL MOVIMIENTO DEL BRAZO
//SE DEJA EN TODO MOMENTO ABIERTA LA PINZA DEL BRAZO
actual[4]=90;
enviar(0x3f8,255);
enviar(0x3f8,4);
enviar(0x3f8,actual[4]);

//ASIGNACION DEL ESTADO DE MOVIMEINTO A TODOS LOS MOTORES QUE DEBAN //MOVERSE
        if ((destino[0]-actual[0])!=0){status0=1;}
        if ((destino[1]-actual[1])!=0){status1=1;}
        if ((destino[2]-actual[2])!=0){status2=1;}
        if ((destino[3]-actual[3])!=0){status3=1;}


        if(ycent<0){
        flag = 0;
        status3=0;}
//DEBIDO A QUE EL MOVIMIENTO DEL BRAZO DEBE  EJECUTARSE JUNTO CON LA //CAPTURA
DE UNA NUEVA IMAGEN Y DE SU PROCESAMIENTO
//EL BRAZO REALIZA SE MUEVE 30 POSICIONES ANTES DE REALIZAR OTRA //DIGITALIZACION Y
REPETIR TODO EL PROCESO DE NUEVO
        for(j=0;j<30; j++){
// MOTOR #0
        if ((destino[0]-actual[0])>0){
                actual[0]=actual[0]+1;
enviar(0x3f8,255);
enviar(0x3f8,0).
enviar(0x3f8,actual[0]);}
                if ((destino[0]-actual[0])<0){
                        actual[0]=actual[0]-1;
enviar(0x3f8,255);
enviar(0x3f8,0);
```

```
enviar(0x3f8,actual[0]);}

for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[0]-actual[0])==0){status0=0;}

// MOTOR #1
        if ((destino[1]-actual[1])>0){
                actual[1]=actual[1]+1;
enviar(0x3f8,255);
enviar(0x3f8,1);
enviar(0x3f8,actual[1]);}
                if ((destino[1]-actual[1])<0){
                        actual[1]--;
enviar(0x3f8,255);
enviar(0x3f8,1);
enviar(0x3f8,actual[1]);}
for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[1]-actual[1])==0){status1=0;}

// MOTOR #2
        if ((destino[2]-actual[2])>0){
                actual[2]++;
enviar(0x3f8,255);
enviar(0x3f8,2);
enviar(0x3f8,actual[2]);}
                if ((destino[2]-actual[2])<0){
                        actual[2]--;
enviar(0x3f8,255);
enviar(0x3f8,2);
enviar(0x3f8,actual[2]);}

                for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[2]-actual[2])==0){status2=0;}

        // MOTOR #3
        if ((destino[3]-actual[3])>0){
                actual[3]++;
enviar(0x3f8,255);
enviar(0x3f8,3);
enviar(0x3f8,actual[3]);}
                if ((destino[3]-actual[3])<0){
        actual[3]--;
enviar(0x3f8,255);
enviar(0x3f8,3);
enviar(0x3f8,actual[3]);}

for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[3]-actual[3])==0){status3=0;}
```

```
}
//EN ESTE PUNTO SE EVALUA SI EL OBJETO HA SALIDO DEL ALCANCE DEL BRAZO
//DEBE RECORDARSE QUE EL VALOR PERMITIDO DEL RADIO ES DESDE 19 HASTA 38
//CENTIMETROS.
        flag =0;
        if((radio>=19)&&(radio<=38)){
                if((pos0>=10)&&(pos0<=245)) {
                        flag =1;}}
//ESTA INSTRUCCIÓN FORMA PARTE DEL DO- WHILE QUE SE ABRIO AL INICIO DEL
//PROGRAMA Y ES EL QUE PERMITE SACAR AL PROGRAMA DE SU BUSQUEDA DE //OBJETOS.
EL SIGUIENTE PASO ES REGRESAR AL BRAZO A SU POSICION INICIAL

}while(flag = =1);

//REGRESO DEL MOTOR A SU POSICION DE DESCANSO SI EL OBJETO SALIO DE SU //ALCANCE

destino[1]=255;
destino[2]=255;
destino[3]=200;


        if ((destino[0]-actual[0])!=0){status0=1;}
        if ((destino[1]-actual[1])!=0){status1=1;}
        if ((destino[2]-actual[2])!=0){status2=1;}
        if ((destino[3]-actual[3])!=0){status3=1;}

        flag = status0+status1+status2+status3;
                if(ycent<0){
        flag = 0;
        status3=0;}

        while (flag>0){

// MOTOR # 0
        if ((destino[0]-actual[0])>0){
                actual[0]=actual[0]+1;
enviar(0x3f8,255);
enviar(0x3f8,0);
enviar(0x3f8,actual[0]);}
                if ((destino[0]-actual[0])<0){
                        actual[0]=actual[0]-1;
enviar(0x3f8,255);
enviar(0x3f8,0);
enviar(0x3f8,actual[0]);}

for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[0]-actual[0])==0){status0=0;}

// MOTOR #1
        if ((destino[1]-actual[1])>0){
                actual[1]=actual[1]+1;
enviar(0x3f8,255);
enviar(0x3f8,1);
enviar(0x3f8,actual[1]);}
```

```
                if ((destino[1]-actual[1])<0){
                        actual[1]--;
enviar(0x3f8,255);
enviar(0x3f8,1);
enviar(0x3f8,actual[1]);}
for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[1]-actual[1])==0){status1=0;}

// MOTOR #2
        if ((destino[2]-actual[2])>0) {
                actual[2]++;
enviar(0x3f8,255);
enviar(0x3f8,2);
enviar(0x3f8,actual[2]);}
                if ((destino[2]-actual[2])<0) {
                        actual[2]--;
enviar(0x3f8,255);
enviar(0x3f8,2);
enviar(0x3f8,actual[2]);}

                for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[2]-actual[2])==0){status2=0;}

// MOTOR #3
        if ((destino[3]-actual[3])>0){
                actual[3]++;
enviar(0x3f8,255);
enviar(0x3f8,3);
enviar(0x3f8,actual[3]);}
                if ((destino[3]-actual[3])<0){
                        actual[3]--;
enviar(0x3f8,255);
enviar(0x3f8,3);
enviar(0x3f8,actual[3]);}

for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[3]-actual[3])==0){status3=0;}

        flag = status0+status1+status2+status3;

}
//MUEVE SOLO AL MOTOR DE LA BASE POR SEPARADO
                destino[0]=130;
        status0=1;
        while(status0>0){
                if ((destino[0]-actual[0])>0){
                actual[0]=actual[0]+1;
enviar(0x3f8,255);
enviar(0x3f8,0);
enviar(0x3f8,actual[0]);}
```

```
                if ((destino[0]-actual[0])<0){
                        actual[0]=actual[0]-1;
enviar(0x3f8,255);
enviar(0x3f8,0);
enviar(0x3f8,actual[0]);}

for (d=0;d<=20;d++){
        for (n=0;n<=20000;n++);}

        if ((destino[0]-actual[0])==0){status0=0;}
        }
                pxd_close( );
         exit ( 0 );
                k++;
}

//SE CREA NO PARA LA APLICACIÓN DE LA VENTANA, SINO PARA PERMITIR
//QUE WINDOWS PUEDA ACCESAR A  LA FUNCION Y SE EJECUTE DE FORMA NORMAL
          LRESULT CALLBACK MainWndProc(
        HWND    hWnd,
        unsigned   message,
        WPARAM          wParam,
        LPARAM          lParam){
        PAINTSTRUCT paints;
        RECT   rects;
        HDC    hDC;

        switch (message) {

          case WM_CHAR:
          case WM_LBUTTONDOWN:
          case WM_RBUTTONDOWN:
                prox_func(hWnd);
                break;

          case WM_DESTROY:
                PostQuitMessage(0);
                break;

          case WM_CREATE:
                break;
          case WM_PAINT:
                hDC = BeginPaint(hWnd, &paints);
                GetWindowRect(hWnd, &rects);
                DrawText(hDC, " Presione una tecla o haga clic", -1, &rects, DT_LEFT|DT_VCENTER);
                EndPaint(hWnd, &paints);
                break;

          default:
                return(DefWindowProc(hWnd, message, wParam, lParam));
        }
        return(0);
  }
```

Los tres programas anteriores se han hecho digitando de manera directa el código, la aplicación que viene a continuación fue hecha por medio de *AppWizard* y se usó para dar una interfaz gráfica de fácil uso (ya que opera con botones) al usuario. Cada botón llama una aplicación: El objeto más a la derecha o a la izquierda, el que se encuentre más cerca o más lejos del brazo, el más grande o uno que presenta las coordenadas calculadas del objeto sin que exista ningún movimiento del brazo. El programa fue modificado por medio de *ClassWizard* para que realizara el llamado de los programas (por medio de _SPAWNLP) que ya se encuentran en formato EXE al momento de presionar cada uno de los botones correspondientes. Como se mencionó antes, se crea un esqueleto de programa de todos las *Clases* necesarias para la aplicación requerida y los programas desarrollados son muchos, por lo que se presenta solamente el archivo CPP de construcción de la ventana de la aplicación.

LISTADO DEL ARCHIVO CPP PRINCIPAL DEL PROGRAMA DE LLAMADO DE TODAS LAS RUTINAS.

```
// padreDlg.cpp : implementation file

#include "stdafx.h"
#include "padre.h"
#include "padreDlg.h"
#include <stdio.h>
#include <process.h>


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
```

```
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPadreDlg dialog

CPadreDlg::CPadreDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CPadreDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CPadreDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CPadreDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CPadreDlg)
	DDX_Control(pDX, IDC_TEGOBMPCTRL1, m_bmp);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPadreDlg, CDialog)
```

```
	//{{AFX_MSG_MAP(CPadreDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_CALIBRAR, OnCalibrar)
	ON_BN_CLICKED(IDC_CERRAR, OnCerrar)
	ON_BN_CLICKED(IDC_AREA, OnArea)
	ON_BN_CLICKED(IDC_SEGUIR, OnSeguir)
	ON_BN_CLICKED(IDC_X_CERCA, OnXCerca)
	ON_BN_CLICKED(IDC_Y_CERCA, OnYCerca)
	ON_BN_CLICKED(IDC_COORDENADAS, OnCoordenadas)
	ON_BN_CLICKED(IDC_X_LEJOS, OnXLejos)
	ON_BN_CLICKED(IDC_Y_LEJOS, OnYLejos)
	ON_BN_CLICKED(IDC_UMBRAL, OnUmbral)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPadreDlg message handlers

BOOL CPadreDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CPadreDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
```

```
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CPadreDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CPadreDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

void CPadreDlg::OnCalibrar()
{
_spawnlp(_P_WAIT,"C:\\Proyecto\\Calib.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

}

void CPadreDlg::OnCerrar()
{
OnOK();
}
```

```
void CPadreDlg::OnArea()
{
_spawnlp(_P_WAIT,"C:\\Proyecto\\Inicio.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

_spawnlp(_P_WAIT,"C:\\Proyecto\\MaArea.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

}

void CPadreDlg::OnSeguir()
{
_spawnlp(_P_WAIT,"C:\\Proyecto\\Seguir1.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");
        // TODO: Add your control notification handler code here

}

void CPadreDlg::OnXCerca()
{
_spawnlp(_P_WAIT,"C:\\Proyecto\\Inicio.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

_spawnlp(_P_WAIT,"C:\\Proyecto\\MenorX.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");
        // TODO: Add your control notification handler code here

}

void CPadreDlg::OnYCerca()
{
_spawnlp(_P_WAIT,"C:\\Proyecto\\Inicio.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

_spawnlp(_P_WAIT,"C:\\Proyecto\\MenorY.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

}

void CPadreDlg::OnCoordenadas()
{
_spawnlp(_P_WAIT,"C:\\Proyecto\\Coorde.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

}

void CPadreDlg::OnXLejos()
{
        _spawnlp(_P_WAIT,"C:\\Proyecto\\Inicio.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

_spawnlp(_P_WAIT,"C:\\Proyecto\\MayorX.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

}
```

```
void CPadreDlg::OnYLejos()
{
        _spawnlp(_P_WAIT,"C:\\Proyecto\\Inicio.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

_spawnlp(_P_WAIT,"C:\\Proyecto\\MayorY.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

}

void CPadreDlg::OnUmbral()
{
        // TODO: Add your control notification handler code here
_spawnlp(_P_WAIT,"C:\\Proyecto\\umbral.exe",NULL);
m_bmp.SetBmpFilename("c:\\imagenes\\cali.bmp");

}
```

Lo antes listado no es el único archivo CPP que se crea sino que son varios, además de los archivos H, los archivos de recursos RSC (donde se encuentran las definiciones de la parte visual de la aplicación y el archivo DSW (que permite ejecutar al *Workspace*), pero es el que controla la creación de la caja de diálogo y donde colocamos nuestro código.

Note en el programa anterior que la función _SPAWNLP hace uso (entre sus argumentos) de la opción _P_WAIT, lo cual permite que el *Workspace* de llamado de las aplicaciones (el programa con botones) espere hasta que la selección haya terminado para continuar con su operación. Esto es necesario para evitar que dos programas hagan uso de la tarjeta digitalizadora al mismo tiempo. Lo especificado entre comillas es la ruta donde se encuentran los programas ejecutables.

La variable que se encuentra inmediatamente debajo de la función ha sido adicionada por medio de *ClassWizard* y se usa para llamar a una ventana de video tipo Direct X que se creó para mejorar la presentación de las imágenes, dentro de esta variable (que realmente es un objeto de Visual C++) se debe de especificar la ruta donde se encuentra el archivo BMP a presentar.