ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Carrera

RECONOCIMIENTO DE IMÁGENES PARA SISTEMAS DE CONTROL DE TIEMPO REAL

Para acceder al Titulo de

INGENIERO DE TELECOMUNICACIÓN

Autor: María Campo-Cossío Gutiérrez

Mayo – 2005

Gracias a mis padres y a mi hermana, por todo su apoyo.

A mis compañeros de carrera por todos esos ratos, en especial a Raúl por toda su ayuda.

A todo departamento de Electrónica y Computadores de la UC, por su ayuda y por habernos hecho pasar tantos buenos momentos y por que sin su ayuda habría sido muy difícil terminar este proyecto, en especial a Michael, por todas las buenas ideas.

Y a Jose Mª Salmón por construir nuestro juguete...

INDICE DE CONTENIDO:

0. RESUMEN:

Este proyecto trata de demostrar el funcionamiento de sistemas de control basados en el procesamiento de imágenes, desde sistema operativo MaRTE, desarrollado en el departamento de Electrónica y Computadores de la Universidad de Cantabria, en una aplicación de tiempo real.

Como aplicación, se pensó en el control en dos dimensiones de una bola sobre un plano accionado por servomotores. El sistema incluiría, como entrada de datos, una cámara analógica, para la que ya existían drivers en MaRTE SO, cuyas imágenes serían digitalizadas a través de una tarjeta digitalizadora (Frame Grabber).

Partiendo de las imágenes capturadas por la videocámara cada 20 ms, con una resolución de 640x480 pixeles, debemos, en primer lugar, encontrar el centro de masas de la bola, a continuación, en base a este centro, los servomotores deben mover la plataforma para que la bola siga una trayectoria determinada. La búsqueda del centro y las tareas de control deben cumplir la citada restricción temporal.

Debido a la complejidad del sistema, se decidió dividir el trabajo en dos proyectos fin de carrera. Este proyecto se centra en el procesado de imágenes, mientras que el proyecto complementario, realizado por Raúl Arnau Prieto, resuelve el sistema de control de los motores. Finalmente ambos proyectos se integran para controlar totalmente el sistema.

El procesado de la imagen se divide en varias etapas. La primera etapa, consiste en un preprocesado de la imagen con el objetivo de mejorar la imagen a analizar. Para ello, se selecciona una ventana de visión a partir de la última posición conocida de la bola. De este modo, el tiempo de procesado de las etapas siguientes se reduce considerablemente y sólo se procesa la imagen completa en la primera iteración del ciclo, o en los casos en los que se desconozca el centro de la bola (debido a errores). A continuación se segmenta la imagen, distinguiendo los pixeles correspondientes al objeto de los que forman parte del resto de la imagen. Para ello se establecen los umbrales que determinan las zonas de la imagen en las que se encuentra la bola, a partir del contenido de color de cada píxel en el histograma de la imagen. Por último se elimina el ruido, o pixeles que en la etapa anterior fueron almacenados como pertenecientes a la bola, mediante una etapa de erosión.

A partir del resultado de la segmentación de la imagen, se calcula el centro de masas de la bola. Posteriormente, se corrige este centro debido a la distorsión radial introducida por la cámara. Esta distorsión, provocada por efectos en la curvatura de las lentes, hace que las lineas rectas de la imagen real, que aparezcan como lineas curvas, con lo cual la información útil que vamos a extraer de la imagen no sea todo lo exacta y real que queremos.

El centro de la bola, servirá como realimentación del sistema de control, que se encargará de colocar los servomotores en la posición óptima para conseguir que la bola alcance las sucesivas posiciones deseadas que forman la trayectoria a seguir en el plano.

La librería de procesado de imagen se desarrolló en primer lugar en Linux, debido a la dificultad para extraer resultados en MaRTE (ya que no soporta entrada y salida de ficheros). Para llevar a cabo las pruebas, se procesaron imágenes sobrecargadas de objetos y colores, comprobándose que la librería era capaz, incluso en estas circunstancias, tanto de cumplir las restricciones temporales, como de realizar correctamente las tareas.

Una vez desarrollada la librería de procesado de imagen y comprobado su funcionamiento y cumplimiento de restricciones temporales, se implementó en MaRTE y se integró con la librería de captura de imágenes y con el sistema de control. Esto nos permitió evaluar las posibilidades de realizar un sistema de control basado en el procesamiento de imagen sobre el sistema operativo MaRTE OS.

1. INTRODUCCIÓN:

1.1 INTRODUCCIÓN A LA VISIÓN POR COMPUTADOR:

Actualmente, es cada vez mayor el número de sistemas de control que se van incorporando a la industria, al sector servicios, etc. Incluso, ya es común encontrarlos en control de accesos, control de tráfico, control industrial, robots, teledetección por satélite, conducción automática (en desarrollo), etc..

Sin embargo, la aplicación de un sistema de control para realizar una determinada tarea depende, en un elevado porcentaje, del conocimiento a *priori* del espacio de trabajo y de la localización de los objetos a manipular. Esta limitación es debida a que los sistemas de control industriales comerciales no integran sistemas sensoriales, que les permitan adaptarse a su entorno. Dentro de los sistemas sensoriales adaptables a los sistemas de control industriales comerciales, los sensores visuales se han convertido en elementos cada vez más frecuentes en las aplicaciones de control recientes. Su principal ventaja es que permiten obtener una descripción del entorno bastante completa de forma no intrusiva. Se podría decir que, en general, facilitan la integración de dispositivos de control en entornos menos estructurados que los que se encuentran en aplicaciones clásicas de automatización.

La interacción entre sensores visuales y sistemas de control ha sido y es objeto de gran interés en la investigación de las últimas décadas. Tiene especial relevancia en aplicaciones en las que un *manipulador robótico* o robot (en el sentido más genérico de *sistema eléctrico-mecánico articulado susceptible de ser accionado*) tiene que interactuar con objetos, bien se trate de objetos estáticos cuya localización no sea conocida de forma precisa o simplemente sea impredecible, o bien de objetos móviles, cuyo perfil de movimiento no sea conocido, como es en nuestro caso.

La expresión *realimentación visual* se aplica a sistemas guiados visualmente, los cuales hacen uso de una o varias cámaras para obtener información en forma de imágenes, que es empleada como señal de realimentación para dirigir o controlar a un elemento durante la realización de una tarea. Cuando se usa realimentación visual, se deben extraer de las imágenes capturadas algunas características del objeto de interés que proporcionen suficiente información para el control.

Basado en esta entrada sensorial, se genera una secuencia de control. Además, el sistema puede también requerir una inicialización automática que comúnmente incluye extracción de características deseadas o/y reconocimiento del objeto.

En términos de manipulación, una de las principales motivaciones para incorporar la visión al lazo de control ha sido la demanda de sistemas de control más flexibles. Un control en lazo cerrado, donde la visión es usada como el principal sensor, consta normalmente de dos procesos entrelazados: realimentación visual y control. La realimentación y extracción de información visual proporciona una estimación y actualización continua de las características durante el movimiento del robot/objeto. La conveniencia del uso del color en el procesamiento de imágenes radica, principalmente, en que el color es un descriptor poderoso que frecuentemente permite simplificar la identificación y extracción de objetos de una escena y, también, en que el ojo humano puede discernir cientos de tonalidades e intensidades de un color, mientras que sólo cerca de dos docenas de tonalidades de gris.

La expansión de las máquinas de visión en colores está garantizada por la disponibilidad de cámaras color CCD (charge-coupled device) de bajo precio, alta resolución y buena calidad. Además, la existencia de microprocesadores con funciones de multimedia que favorecen el procesamiento de imágenes (ej.: Pentium MMX) la continua disminución del precio de la memoria (ej.: memorias SD RAM), los buses de interface con elevadas tasas de transferencia (ej.: el bus PCI) los sistemas operativos de tiempo real y el conocimiento de técnicas de análisis y algoritmos optimizados, son otros factores que pueden contribuir a la difusión de los sistemas de visión en colores.

Aunque muchos sistemas de visión comerciales presentan pocas o nulas funciones para el tratamiento de imágenes en colores, es posible aplicar todas estas técnicas y herramientas a cada uno de los planos de color que conforman la imagen, como si se tratara de una imagen monocromática.

El control visual de manipuladores robóticos *Visual Servoing* ha sido estudiado durante más de tres décadas. Más recientemente, esta área ha atraído una significativa atención debido a que los avances computacionales han hecho posible su implementación de tiempo real.

1.2 MOTIVACIÓN Y OBJETIVO DEL PROYECTO:

La principal motivación de este proyecto surge de la escasa cantidad de aplicaciones de tiempo real diseñadas específicamente para Marte SO y de la necesidad de evaluar la respuesta y buen funcionamiento del mismo ante aplicaciones de tiempo real.

Por otra parte, la integración de la capturadora de imágenes es aun muy reciente y son escasas las aplicaciones dotadas de visión sobre Marte. Las aplicaciones basadas en el procesado de imágenes tienen cada vez más relevancia en el campo de la automatización industrial, para el que está especialmente pensado MaRTE.

El objetivo principal sería por tanto estudiar y comprobar tanto las posibilidades y limitaciones del SO en una aplicación de control basado en el procesamiento de imagen con restricciones de tiempo real, como de la librería de captura de imágenes. Con este objetivo, se pensó en diseñar un sistema integrado por varias partes; una plataforma accionada por servomotores con una cámara fija, tratamiento de imágenes y control de los motores.

La complejidad del sistema nos llevó a dividir el trabajo en dos partes. El objetivo fundamental de este proyecto en concreto, es el desarrollo de una librería de tratamiento de imágenes capaz de encontrar el centro de masas de un objeto a partir de una imagen, con la mayor exactitud posible, ya que servirá como realimentación del algoritmo de control. Además, debe cumplir las restricciones temporales que permitan al sistema de control realizar correctamente su tarea.

Como objetivo secundario, la librería desarrollada en este proyecto, podrá ser reutilizable en aplicaciones futuras, dejando para ello opciones de uso para el programador. Además, debido a que previamente a la aplicación en MaRTE de la librería será necesario comprobar su funcionamiento en Linux, tendrá una versión para este sistema operativo, nuevamente con varias opciones para ser aplicable en el mayor número de casos.

Una vez desarrolladas las librerías y comprobado su buen funcionamiento, el siguiente objetivo será su integración con la librería bttv_marte de captura de imágenes y con el sistema de control, para posteriormente comprobar su funcionamiento conjunto bajo MaRTE.

1.3 ESTRUCTURA DEL DOCUMENTO:

El documento esta dividido en 7 capítulos, en el primero se recogen aspectos esenciales como son la motivación y los objetivos de este proyecto .En el capítulo 2 se presentan los fundamentos teóricos de sistemas operativos de tiempo real, así como una breve descripción del SO MaRTE.

El tercer capítulo describe los elementos que componen el sistema y sus características, así como el modo en que se intercambian datos entre ellos. Describe así mismo las características fundamentales de las imágenes y de las librerías utilizadas.

El cuarto capítulo describe el proceso de calibración de la cámara analógica, mediante el cual se trata de corregir la distorsión radial que ésta introduce. A causa de este efecto las líneas rectas de la imagen real aparecen como curvas.

El quinto capítulo se centra en el procesado de imagen, comienza con una breve introducción de aspectos teóricos y operaciones matemáticas, así como de los pasos necesarios en el procesado de la imagen. A continuación, se presentan los resultados obtenidos para imágenes sobrecargadas (llenas de objetos y colores) y para imágenes reales del sistema, así como los tiempos de ejecución de las tareas para las aplicaciones de Linux y MaRTE. Se describen también en este capítulo las librerías desarrolladas.

El sexto capítulo trata de la integración del sistema completo, el intercambio de datos entre todas las tareas, y los tiempos de ejecución medidos en el sistema.

Finalmente, en el séptimo capítulo se resumen los resultados obtenidos.

2. SISTEMAS OPERATIVOS DE TIEMPO REAL:

Un sistema de tiempo real es una combinación de uno o varios computadores, dispositivos hardware de entrada / salida, y software de propósito especial, que interaccionan con el medio que les rodea [GONM2001]. Debido a la naturaleza cambiante del entorno con el que interactúan, junto con la necesidad de coordinación entre sus componentes, la validez de los resultados no depende solo de que la lógica e implementación de los programas sean correctos, sino también del tiempo en el que dicha operación entregó su resultado. Si las restricciones de tiempo no son respetadas, se dice que el sistema ha fallado.

Por lo tanto, es esencial que las restricciones de tiempo en los sistemas sean cumplidas. El garantizar el comportamiento en el tiempo requerido necesita que el sistema sea predecible, por lo tanto los tiempos de respuesta deben ser acotados [ALDM2002].

En los sistemas de tiempo real se requieren métodos y herramientas fiables, que garanticen los requisitos temporales y que permitan utilizar plataformas en las que realizar pruebas y medir los tiempos con precisión. Los sistemas convencionales no tienen un comportamiento temporal predecible, es decir, no permiten garantizar los tiempos de respuesta y no acotados.

2.1 TIPOS DE SISTEMAS DE TIEMPO REAL:

Podemos clasificar los sistemas de tiempo real como:

- Sistemas de tiempo real estricto (Hard Real Time):

Cuando el sistema debe responder siempre a los eventos de una forma determinista, con tiempo acotado. El no cumplimiento puede tener consecuencias más o menos graves y en algunos casos puede ser preferible un trabajo imperfecto pero terminado a tiempo. Un ejemplo sería el control electrónico de estabilidad de Mercedes-Benz (ESP), que debe asegurar la respuesta dentro de un tiempo acotado o de lo contrario la acción de control sería inútil.

- Sistemas de tiempo real no estricto (Soft Real Time):

Cuando el sistema debe responder de una forma determinista, pero las restricciones temporales son suaves siendo suficiente que el comportamiento del sistema esté dentro de un rango de tolerancia. Por ejemplo, un sistema de apertura de una puerta.

- Sistemas de tiempo real firme (Firm Real Time):

Son sistemas de tiempo real estricto pero pueden tolerar pérdidas, si la probabilidad de ocurrencia de las mismas es baja. Por ejemplo la pérdida de una trama de audio o vídeo.

Para implementar un sistema del primer tipo, necesitaremos un sistema operativo que garantice la ejecución del código generado con restricciones de tiempo real.

2.2 CARACTERÍSTICAS DE LOS SISTEMAS OPERATIVOS DE TIEMPO REAL:

Se caracterizan por presentar requisitos especiales en cinco áreas generales:

- Predecibilidad
- Latencia
- Planificabilidad
- Fiabilidad
- Tolerancia a los fallos

Una característica distintiva de un sistema de tiempo real es la predecibilidad. Implica que debe ser posible demostrar o comprobar a priori que los requerimientos de tiempos se cumplen en cualquier circunstancia. Como consecuencia, la predecibilidad implica:

- Una cuidadosa planificación de tareas y recursos.
- Cumplimiento predecible de requisitos temporales: determinismo.
- Anticipación a fallos, y sus requerimientos temporales.
- Consideraciones de sobrecargas: degradación controlada.
- Consideraciones de elementos de impredecibilidad.
- Dotar al sistema con capacidades de monitorización y control de tiempos (hardware, software, sistema operativo, lenguaje, lineas y protocolos de comunicaciones).

La **latencia** se refiere a cuanto tiempo consume un sistema operativo en dar servicio a la interrupción después de reconocerla. Las características de la latencia, entre otras:

1- La cantidad de tiempo necesario para iniciar la gestión de interrupciones, cambios de contexto y a servicios bloqueantes.

2- La cantidad de tiempo necesario para ejecutar la gestión. Generalmente, depende de la plataforma del hardware y el retraso del bus.

En sistema operativo típico que no sea de tiempo real, el usuario no tiene control sobre la función de **planificación** del sistema operativo. En un sistema de tiempo real resulta esencial permitir al usuario un control preciso sobre la prioridad de las tareas. El usuario debe poder distinguir entre tareas rígidas y flexibles y especificar prioridades relativas dentro de cada clase. Un sistema de tiempo real también permitirá al usuario especificar qué procesos deben estar siempre residente en la memoria principal.

Un tipo de planificación común, consiste en asignar prioridades a los procesos, en cada momento, la CPU se cede al trabajo más prioritario. Esta prioridad puede ser asignada permanentemente, o bien puede variar, por ejemplo en función de información nueva que se tenga sobre el comportamiento de un proceso.

La **fiabilidad** es normalmente mucho mas importante en sistemas de tiempo real que en los que no lo son. Un fallo transitorio en un sistema que no sea de tiempo real puede resolverse simplemente volviendo a reiniciar el sistema.

Un fallo de un procesador en un multiprocesador que no sea de tiempo real produce una reducción del nivel de servicio hasta que se repara o sustituye el procesador averiado. Pero un sistema de tiempo real responde y controla sucesos en tiempo real. Las perdidas o degradaciones del rendimiento pueden tener consecuencias catastróficas, que pueden ir desde perdida financieras hasta daños en equipo e incluso perdida de vidas humanas.

La **tolerancia a los fallos** es una característica que hace referencia a la capacidad de un sistema de conservar la máxima capacidad y los máximos datos posibles en caso de fallos. Un sistema de tiempo real intentara corregir el problema o minimizar sus efectos mientras continua la ejecución (no todos los sistemas operativos de tiempo real cumplen este requisito, pero sí aquellos sistemas críticos).

Un aspecto importante a la tolerancia a los fallos es la **estabilidad**. Un sistema de tiempo real es estable si, en los casos en los que es imposible cumplir todos los plazos de ejecución de las tareas, el sistema cumple los plazos de las tareas mas criticas y de mayor prioridad, incluso si no se cumplen los de alguna tarea menos critica.

Para cumplir los requisitos anteriores los sistemas operativos actuales de tiempo real incluyen normalmente las siguientes características:

- Cambios rápidos de procesos o hilos.
- Pequeño tamaño (con una mínima funcionalidad asociada).
- Capacidad de responder rápidamente a interrupciones externas.
- Multitarea con herramientas de comunicación entre procesos, como semáforos, señales y sucesos.
- Planificación preferente basadas en prioridades.

- Reducción de intervalos en los que están inhabilitadas las interrupciones.
- Primitivas para demorar tareas durante un tiempo fijo y para detenerlas y reanudarlas.
- Alarmas especiales y temporizadores de alta resolución.
- Gestión de memoria con tiempos de respuesta predecibles
- Tiempos de respuesta acotados en todos los servicios.

El corazón de un sistema de tiempo real es el planificador de tareas a corto plazo. Lo que resulta importante es que todas las tareas de tiempo real estricto acaben (o comiencen) en su plazo y que también acaben (o comiencen) en su plazo tantas tareas flexibles de tiempo real como sea posible.

La mayoría de los sistemas operativos actuales de tiempo real son incapaces de trabajar directamente con plazos. En su lugar, se han diseñado para ser tan sensibles como sea posible a las tareas de tiempo real, de forma que, cuando se aproxima un plazo se pueda planificar rápidamente. Las aplicaciones de tiempo real normalmente necesitan tiempos de respuesta acotado en un rango de varios milisegundos, pero hay sistemas, como por ejemplo el que se ha desarrollado en este proyecto, presentan restricciones en un rango de diez a cien microsegundos.

2.2 SISTEMA OPERATIVO MARTE:

MaRTE OS (Minimal Real-Time Operating System for Embedded Applications) [1], es un núcleo de tiempo real para aplicaciones empotradas que sigue el perfil de sistema de tiempo real mínimo definido en el estándar POSIX.13.

POSIX (Portable Operating System Interface), es una de las normas internacionales de estandarización cuyo objetivo es permitir la portabilidad de aplicaciones a nivel de código fuente entre diferentes sistemas operativos.

Con este fin, y basándose en la interfaz del extensamente utilizado sistema operativo Unix, el estándar POSIX define la interfaz que los sistemas operativos deben ofrecer a las aplicaciones, así como la semántica de los servicios ofrecidos por esa interfaz.

La interfaz descrita en el estándar permite escribir aplicaciones de tiempo real de forma portable. Sin embargo, debido al tamaño del estándar POSIX, resulta inabordable su implementación completa en un sistema operativo destinado a computadores empotrados pequeños. Por esta razón en el estándar POSIX.13 se define el “Sistema de Tiempo Real Mínimo”, que está pensado para aplicaciones empotradas pequeñas. Dicho perfil incluye un pequeño subconjunto de toda la funcionalidad definida en el estándar POSIX, lo que permite su implementación en un núcleo de sistema operativo pequeño y eficiente.

El sistema operativo MaRTE OS está diseñado conforme con el perfil mínimo. Su estructura interna es modular y conocida de forma que, es sencillo incorporar nuevos servicios para proceder a su prueba y evaluación. Es importante destacar la fiabilidad, MaRTE OS se encuentra escrito en su mayor parte en lenguaje Ada (más una pequeña parte en lenguaje C y ensamblador del mc68332), con lo que se potencia la fiabilidad del código generado. MaRTE OS constituye una experiencia pionera, al ser uno de los primeros sistemas operativos escritos en Ada que permite la ejecución de aplicaciones concurrentes escritas en C, Ada, o una mezcla de ambos.

El sistema operativo MaRTE OS no consiste únicamente en un núcleo en el que se implementa la funcionalidad incluida en el subconjunto mínimo del POSIX, sino que además consta de una serie de aplicaciones y servicios que posibilitan la creación, carga y depuración de las aplicaciones. Ha sido necesario, por tanto, crear un entorno de desarrollo cruzado basado en Linux y en los compiladores de GNU GCC y GNAT.

2.3.1 ENTORNO DE DESARROLLO CRUZADO:

Un entorno de desarrollo cruzado estará formado por una parte hardware y otra software, en la parte hardware encontramos el sistema de desarrollo ('host') máquina que tendrá instalada el compilador y depurador cruzados, teniendo como misión generar código para la plataforma de ejecución ('target').

El sistema de desarrollo será una máquina que trabaje con un sistema operativo (Linux, en nuestro caso) que soporte las herramientas necesarias para poder generar el código para la plataforma de desarrollo. Herramientas tales como compilador de ada *gnat*, o el compilador *gcc*. La idea será compilar código en lenguaje Ada o C.

Es decir, un compilador cruzado y depurador cruzados con los que podemos compilar una aplicación escrita en lenguaje de alto nivel (Ada o C) o en lenguaje ensamblador, y enlazarla con las librerías necesarias, para la obtención de un archivo listo para ejecutar en la plataforma de desarrollo.

El depurador cruzado servirá para corregir los errores (tanto de compilación como de ejecución) y poder continuar con la depuración sin necesidad de compilar, enlazar y descargar el programa o aplicación otra vez. El depurador cruzado servirá para corregir los errores de ejecución) y poder continuar con la depuración sin necesidad de compilar, enlazar y descargar el programa o aplicación otra vez.

3. ELEMENTOS DEL SISTEMA:

3.1 ESTRUCTURA DEL CONJUNTO:

El diseño de la plataforma se basó en un juguete de madera, pensado para ser utilizado manualmente. Ya que la idea era que el movimiento fuese proporcionado por servomotores, se diseñó en una estructura de aluminio, para aligerar peso. La desventaja del aluminio, es que no es rígido, el movimiento de los motores puede hacer vibrar al conjunto, por este motivo, se opto por colocar refuerzos triangulares, que aportasen firmeza y evitar así que tal vibración se trasladase a la cámara.

El sistema se basa en un plano, encajado en dos marcos, cada uno de ellos movido por un servomotor (Fig 3.1), consiguiéndose así movimiento en dos dimensiones. Los servomotores se colocaron en el centro de cada plano para aprovechar al máximo su rango de movimiento, y serán controlados a través del puerto paralelo de la plataforma de ejecución.

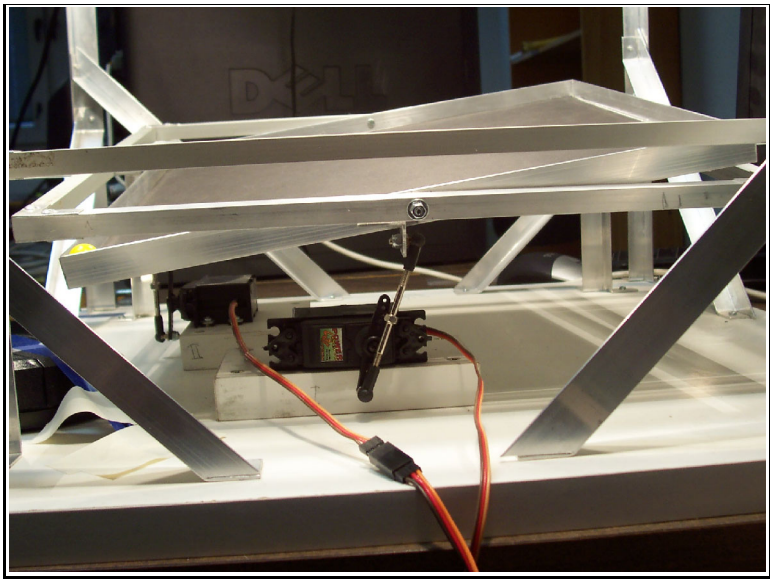


Fig. 3.1 Plano y servomotores

Las primeras imágenes tomadas, mostraron que el sistema necesitaba estar bien iluminado, ya que con poca luz, la imagen pierde gran parte del contenido en color. Además, las sombras que aparecen alrededor de la bola dificultan el proceso de detección.

Sin embargo, el fondo del plano es metálico, el reflejo que se produce al iluminar es tal, que la capturadora se satura, solo veríamos luz. Es por ello que se decidió recubrir el fondo con una superficie opaca y de mínimo rozamiento.

Aun así, los bordes del plano reflejan gran cantidad de luz. El problema reside en que esta luz reflejada, será codificada como “blanco” y coincidirá con el efecto que se produce en las partes de la bola que también reflejen luz. Para evitar este efecto, llevaremos a cabo un exhaustivo procesado de la imagen. Además, dado que la capturadora es muy dependiente de la luz externa que recibe, colocaremos un regulador de luz, de modo que podamos controlar la cantidad de luz que emiten los focos y buscar el punto óptimo, aún cuando las condiciones ambientales varíen.

La figura 3.2 muestra el sistema completo:

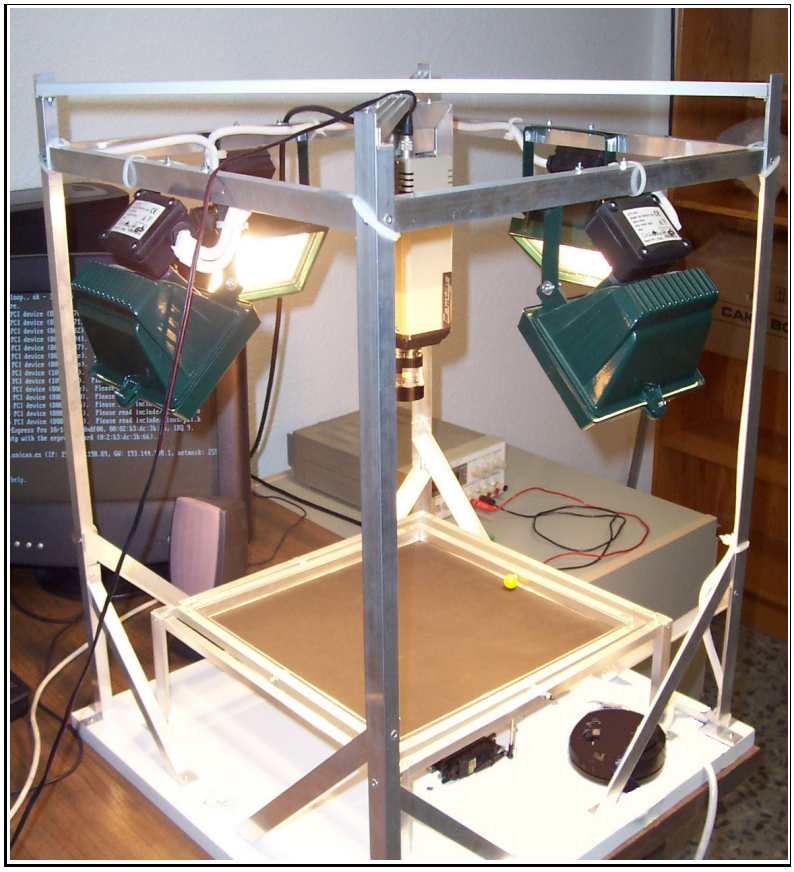


Fig. 3.2 Elementos del conjunto

A grandes rasgos, el sistema completo funciona como sigue: La cámara analógica captura imágenes que serán digitalizadas por una tarjeta digitalizadora. Las imágenes digitalizadas son procesadas para obtener el centro de la bola, en base al cual el sistema de control resuelve la posición necesaria en los motores para conseguir que la bola siga la trayectoria deseada.

Finalmente, esta posición se traslada mediante pulsos a los servomotores a través del puerto paralelo de la plataforma de ejecución.

A grandes rasgos, los distintos elementos del sistema interactúan como se muestra en el siguiente esquema:

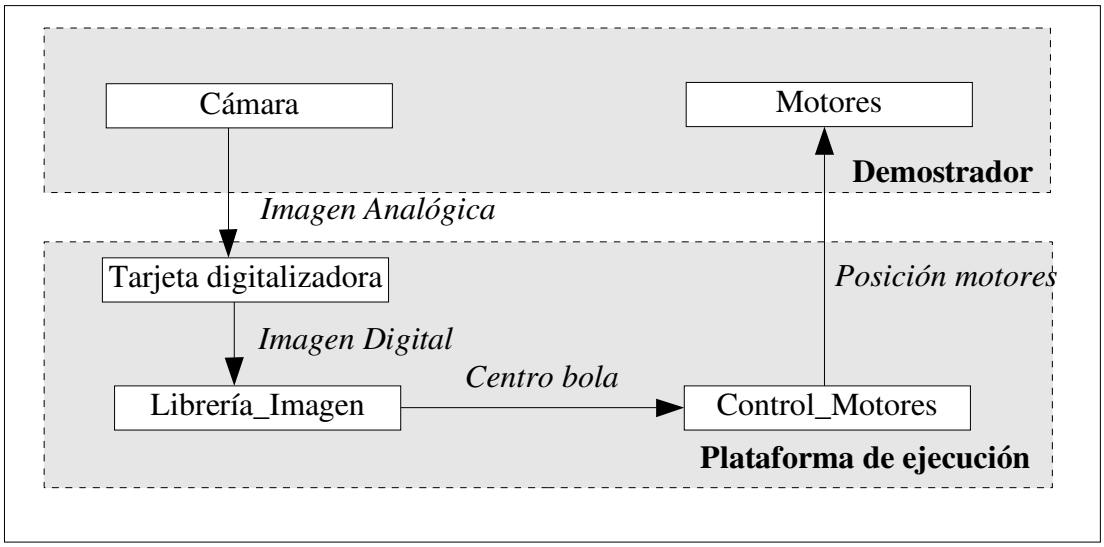


Fig. 3.3 Intercambio de datos

3.1.1 CAPTURA DE IMÁGENES:

3.1.1.1. SEÑAL DE VÍDEO ANALÓGICA:

La captura de imágenes se lleva a cabo mediante una cámara analógica *Cemtys.* La cámara produce una señal compuesta de vídeo analógico, que es digitalizada a continuación a través de una tarjeta digitalizadora (Frame Grabber), *Fusion 878A.* La necesidad de digitalizar las imágenes capturadas por la cámara reside en la naturaleza continua de las señal analógica devuelta.

3.1.1.2. PROCESO DE EXPLORACIÓN DE LA IMAGEN.

En los orígenes de la televisión, se utilizaba un barrido progresivo (todas las líneas de la imagen se barren consecutivamente). Por razones de orden práctico (radiaciones debidas a fugas magnéticas de los transformadores de alimentación, filtrados imperfectos), fue indispensable

utilizar una frecuencia de imagen que estuviera relacionada con la frecuencia de la red (60 Hz en EE.UU., 50 Hz en Europa) para minimizar el efecto visual de estas imperfecciones; la frecuencia de exploración fue, por tanto, de 30 imágenes/s en EE.UU. y de 25 imágenes/s en Europa [PERZ2002]. Estas primeras imágenes presentaban un parpadeo bastante molesto (también llamado flicker de campo).

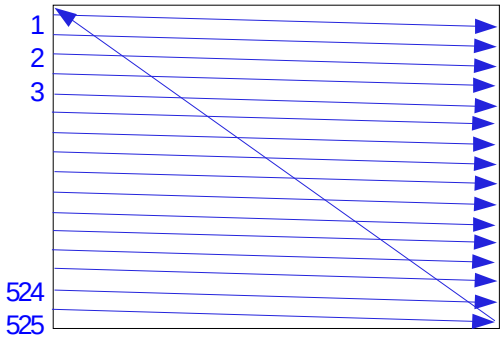


Fig. 3.4 Barrido no entrelazado.

Para solucionar este inconveniente, se utiliza el Barrido entrelazado. Consiste en la transmisión de un primer campo compuesto por las líneas impares de la imagen y a continuación un segundo campo formado por las líneas pares, formándose "media imagen" cada vez, como se muestra en la Figura 3.5. El primer campo comienza con una línea completa y finaliza con media línea, el segundo campo comienza con media línea y finaliza con una línea completa:

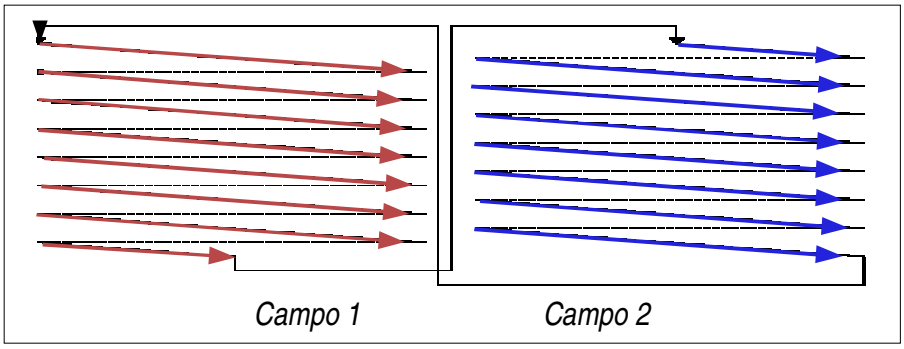


Fig. 3.5 Barrido entrelazado

Posteriormente ambos campos son entrelazados para formar la imagen. Esta forma de barrer la imagen, permite duplicar la frecuencia de refresco de la pantalla (50 o 60 Hz, en lugar de los 25 o 30 Hz) sin aumentar el ancho de banda para un número de líneas dado.

En las aplicaciones de visión artificial, los objetos frecuentemente pasan con cierta velocidad por delante de la cámara. Cuando se utilizan cámaras entrelazadas, los objetos se han movido entre la captura del campo impar y la del campo par, el resultado es que la imagen aparece desenfocada y como si tuviera una doble exposición .

Al reconstruir la imagen y entrelazar los campos, observamos el efecto mostrado en la Fig. 3.6, tomada en nuestro sistema con la bola moviéndose a una velocidad similar a la que alcanza en la aplicación final:

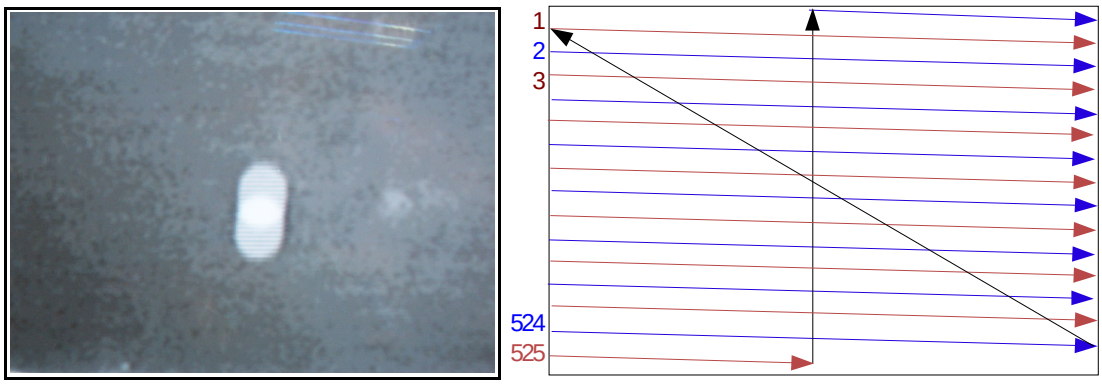


Fig. 3.6 Efecto del barrido entrelazado

Para paliar este efecto algunas cámaras entrelazadas pueden configurarse para leer solo un campo. La resolución vertical en este caso es la mitad, pero la velocidad de captura de imagen se aumenta al doble (50 en el caso de CCIR o PAL). Esta forma de funcionamiento de la cámara se denomina modo campo o modo no entrelazado, y es muy útil en muchas aplicaciones de visión.

3.1.1.3 FRAME GRABBER:

Una vez que la imagen ha sido recogida por la cámara de vídeo, la tarjeta de adquisición y procesamiento de imágenes recibe la señal analógica enviada por ella, para convertirla en una señal digital. Los digitalizadores de imágenes o frame grabbers [DELA2001] muestrean la señal de la cámara y mediante el bus PCI pueden transferir al procesador del ordenador las imágenes digitales línea a línea a gran velocidad y guardan la imagen digital en una zona de memoria, para que pueda ser accedida por el microprocesador del computador para su tratamiento.

El proceso de digitalización se inicia con un proceso de muestreo de la señal. De esta primera parte dependerá en buena medida la calidad final, ya que cuanto más aproximada sea la muestra, más cercana será la imagen final al original. La imagen continua 2D a (x,y) se divide en N filas y M columnas. La intersección de una fila y una columna es un píxel.

El siguiente paso en el proceso es la cuantificación de las muestras recogidas, es decir, asociar un valor al dato recogido en la operación de muestreo, que luego se utilizará en la siguiente fase. La tercera y última fase del proceso de digitalización de una señal es la codificación. En esta fase se ordenan todos los valores que hemos asignado en la fase de cuantificación de una manera concreta.

La tarjeta utilizada en este proyecto es una *Fusion 878A,* que decodifica vídeo NTSC/PAL/SECAM y soporta una resolución de hasta 768 x 576 pixeles. Es capaz de colocar datos de vídeo directamente en memoria, toma el control del bus PCI tan pronto como este disponible y puede transferir datos a una velocidad de 20 Mbytes/s.

3.1.2 SERVOMOTORES:

Los servomotores de radiocontrol se caracterizan por su capacidad para posicionarse de forma rápida en cualquier posición dentro de su intervalo de operación (unos 180º). La conexión al exterior se realiza a través de tres cables; tierra (negro), alimentación (rojo) y señal de control (naranja). El servomotor internamente realiza un control de posición en bucle cerrado, para lo que utiliza un potenciómetro y un circuito de control.

El circuito de control recibe información del movimiento a realizar a través de la línea de control , y de la posición actual a través del potenciómetro. Con esta información, sitúa el eje del motor en su nueva posición.

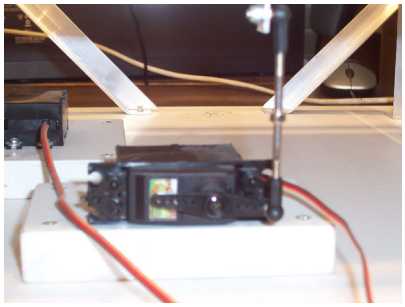


Fig 3.6 Servomotor utilizado

La ***modulación por anchura de pulso***, PWM (*Pulse Width Modulation*), es una de los sistemas más empleados para el control de servos. Este sistema consiste en generar una onda cuadrada en la que se varía el tiempo que el pulso está a nivel alto, manteniendo el mismo período (normalmente), con el objetivo de modificar la posición del servo según se desee.

La duración del pulso indica el ángulo de giro del motor (ver Fig 3.7.a y 3.7.b). Cada servo tiene sus márgenes de operación, que se corresponden con el ancho del pulso máximo y mínimo que el servo entiende. Si el intervalo entre pulso y pulso es inferior al mínimo, puede interferir con la temporización interna del servo, causando un zumbido, y la vibración del brazo de salida. Si es mayor que el máximo, entonces el servo pasará a estado dormido, entre pulsos. Esto provoca que se mueva con intervalos pequeños.

Tras calibrar los servomotores de nuestro sistema con ayuda de un osciloscopio, encontramos que uno de ellos tiene un rango de [0.8, 2.2] ms, mientras que el segundo de [0.73, 2.05] ms. Estos valores serán utilizados por el algoritmo de control a la hora de traducir los ángulos calculados para los motores a pulsos en el puerto paralelo.

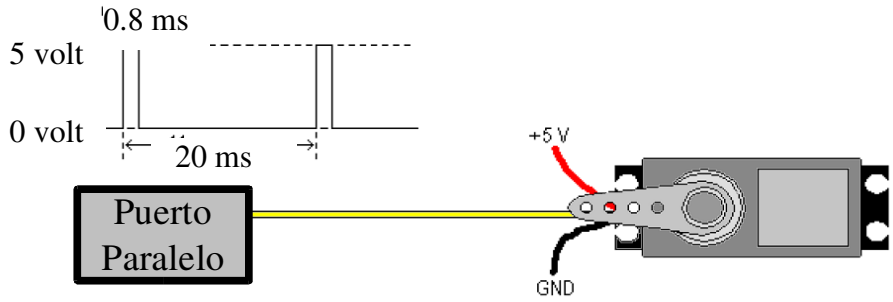


Fig. 3.7.a Servomotor en posición mínima

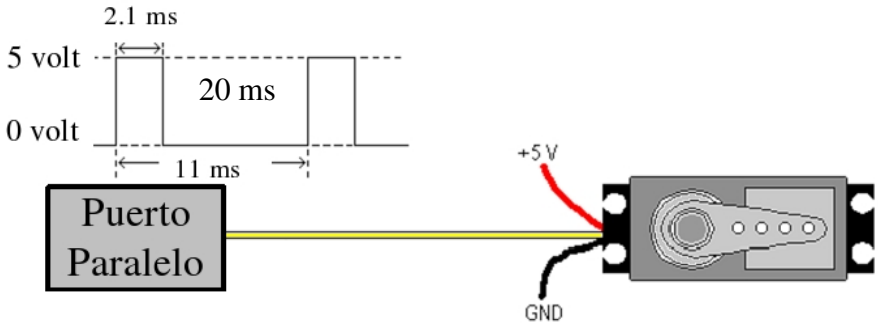


Fig. 3.7.b Servomotor en posición máxima

Es importante destacar que para que un servo se mantenga en la misma posición durante un cierto tiempo, es necesario enviarle continuamente el pulso correspondiente. De este modo, si existe alguna fuerza que le obligue a abandonar esta posición, intentará resistirse. Si se deja de enviar pulsos (o el intervalo entre pulsos es mayor del máximo) entonces el servo perderá fuerza y dejará de intentar mantener su posición, de modo que cualquier fuerza externa podría desplazarlo.

3.1.3 EL PUERTO PARALELO:

Como elemento de conexión entre la plataforma de ejecución y los servomotores se consideró el puerto paralelo o puerto de la impresora. Dicho puerto está diseñado para suministrar datos desde el PC a otros periféricos utilizando como unidad fundamental el byte. Recibe este nombre debido a que envía y recibe datos en forma paralela (al mismo tiempo) 8 bits (1 ó 0), por medio de 8 cables individuales y por tanto es más rápido que el puerto serie al transferir varios datos al mismo tiempo.

Para la comunicación, se utilizan dos lineas de datos, que utilizan loica positiva, es decir, la señal TTL tiene unos 5 voltios cuando esta activa, que es precisamente lo que necesitan leer los motores para funcionar.

Conectaremos las señales de control de los servomotores a dos de las líneas de datos del puerto paralelo y la tierra de ambos servos (es conveniente que compartan la misma tierra para eliminar ruido), a una de las líneas de tierra del puerto paralelo.

La desventaja del puerto paralelo es que el cable no puede ser tan largo como en el puerto serie, ya que este último envía señales de hasta 25 voltios frente a los 5 voltios del paralelo. Sin embargo en nuestra aplicación, la distancia no será un problema.

3.2 PARÁMETROS DE LA IMAGEN:

3.2.1 IMÁGENES:

Una imagen es una función bidimensional de la luz y la intensidad [ADWE1996]. Se representa como f(x,y), para cada punto de la imagen (x,y), la función devuelve el valor de la intensidad de la imagen en dicho punto.

Una imagen [ADWE1996] representa una forma de energía, por tanto ha de ser finita y estrictamente mayor que cero, teniéndose por tanto que f(x,y) toma valores reales enteros en el rango $0 \leq f(x,y) < \infty$.

La diferencia entre una imagen analógica y una digital, está en que la primera de ellas es de naturaleza continua y la segunda de ellas es de naturaleza discreta. Esto quiere decir que una señal digital se representa mediante un número concreto de valores mientras que la representación de una señal analógica se hace a través de una función de infinitos puntos. Por este motivo, la ***digitalización*** de una imagen es una mera aproximación a la señal inicial. (Ver apartado 3.1.1.3)

Las imágenes digitales están formadas por una matriz de elementos (pixeles), cada uno conteniendo el valor de color o intensidad asociado a cada elemento de la matriz, según sus coordenadas espaciales. El origen de coordenadas de una imagen se establece en la esquina superior izquierda.

3.2.2 RESOLUCIÓN, PIXELES Y RELACIÓN DE ASPECTO:

Las imágenes digitales están compuestas por miles o millones de puntos llamados ***pixeles***, cuyo nombre viene de la contracción de “Picture Element" (Elemento de Imagen), y representa la porción más pequeña de la imagen que puede ser manipulada directamente. Cada uno tiene características propias, como lo son el color y la intensidad.

La cantidad de pixeles que contiene una imagen recibe el nombre de ***resolución*** [ADWE1996], y usualmente se refiere esto como un par de números que referencian los ejes horizontal y vertical, es decir, una resolución de 800x600, indica que la imagen podrá ser representada por una matriz de 800 puntos horizontales y 600 verticales, lo cual da un total de 480,000 puntos para procesar y representar una imagen, por lo que, a mayor resolución, mayor cantidad de puntos, mayor la calidad de la imagen y por consiguiente, mayor la cantidad de puntos a procesar en el controlador, mostrar en la pantalla y almacenar en memoria en un momento dado.

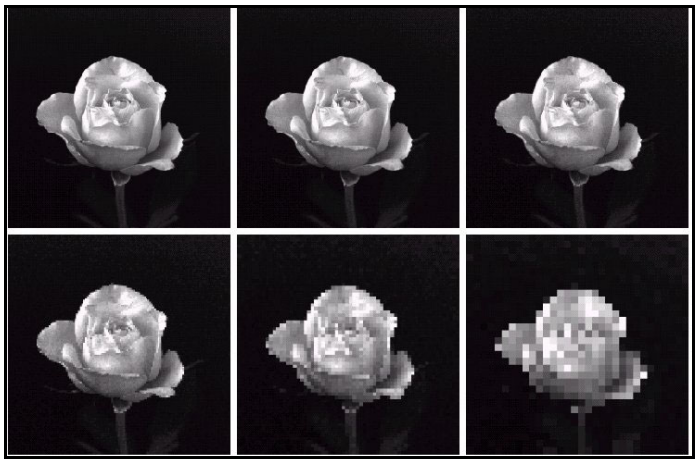


Fig. 3.8 Efecto de la resolución

a) 1024x1024. b) 512x512, c)256x256, d)128x128,e)64x64, f)32x32 pixeles

La ***relación de aspecto*** (aspect ratio) de la imagen es la relación que existe entre la cantidad de pixeles en el eje horizontal y aquellos del vertical [ADWE1996]. La relación de aspecto más común para PCs es 4:3, razón por la cual los monitores son calibrados a esta relación para que las imágenes geométricas no aparezcan desproporcionadas.

3.2.3 PROFUNDIDAD DE PÍXEL:

Este concepto también se le conoce con el nombre de *resolución de bits* y proporciona una medida del número de bits de información que puede almacenar el píxel. Es decir, nos ofrece cuánta información sobre el color puede proporcionarnos cada píxel de la imagen. Evidentemente, a mayor profundidad de píxel tendremos más colores y una más fiel representación de los mismos y por tanto de la imagen. Un píxel con profundidad 1 tiene dos valores posibles: sí/no, blanco/negro (en definitiva, 1 ó 0). Un píxel con profundidad 8 tiene 256 valores posibles, como ocurre con las imágenes en escala de grises o color indexado (256 colores). Un píxel con profundidad 24 tiene 16.000.000 valores posibles que son las imágenes representadas en millones de colores.

3.2.4 COLOR:

Entre los principales tipos de imagen podemos distinguir: blanco y negro, escala de grises, color indexado 16, color indexado 256, color real RGB (los colores se forman mediante la combinación de rojo, verde y azul de forma aditiva) y color real CMAN (consigue los colores a partir de cian, magenta, amarillo y negro). Los tipos de datos se diferencian por la resolución en bits y por el número de canales que comprende la imagen.

- Blanco y negro

Cada píxel de una imagen en blanco y negro es blanco o negro. Los tonos intermedios (grises) se crean ordenando los pixeles blancos y negros para simular gris. Es lo que se llama técnica de 'dithering'.

- Escala de grises

Cada píxel de una imagen en escala de grises puede ser uno de los 256 valores distintos de gris, del negro (cero) al blanco (255), es decir, cada píxel puede ser guardado en un Byte. Este tipo de datos muestra suaves cambios de tono utilizando tonos intermedios de gris. La resolución de una imagen en escala de grises determina el tamaño de los pixeles y, en consecuencia, el número de pixeles de una imagen. Cuanto mayor sea la resolución, los cambios de tono de gris serán más suaves y por tanto más exacta la representación de la imagen (las imágenes de alta resolución también utilizan más memoria).

Los bits de profundidad de color marcan cuántos bits de información disponemos para definir los colores derivados de éstos colores primarios. A más bits, mayor número de variaciones de un color primario podemos tener, es decir, mayor cantidad de colores distintos. Para 256 colores se precisan 8 bits, para obtener miles de colores necesitamos 16 bits (color de alta densidad) y para obtener millones de colores hacen falta 24 bits (color verdadero). Evidentemente, las imágenes en color aportan más información que las imágenes en blanco y negro, ya que permiten trabajar con mas información.

3.2.4.1 MODELOS CROMÁTICOS DE REPRESENTACIÓN DEL COLOR:

El propósito de un modelo de color es facilitar la especificación de colores en alguna forma estándar [ENAL2000]. En esencia, un modelo de color es una especificación de un sistema coordenado de tres dimensiones y de un subespacio dentro de tal sistema, donde cada color se representa por un punto. Dada la simplicidad del espacio tridimensional, éste ha sido escogido como instrumento de representación del color en detrimento de la representación frecuencial.

Entre los modelos orientados al hardware se encuentran: el modelo RGB (Red, Green, Blue) utilizado en monitores y cámaras de vídeo a colores; el modelo CMY (Cyan, Magenta, Yellow) usado en impresoras; y el modelo YIQ (luminosidad, Inphase, Quadrature) que es el estándar para la transmisión de señal de televisión en formato NTSC (National Television Standard Committee).

Entre los modelos de color para la manipulación de imágenes (tales como la creación de gráficos en color para animaciones) se encuentran los modelos HSI (Hue, Saturation, Intensity) y HSV (Hue, Saturation, Value).

Describimos en el siguiente apartado el modelo RGB, ya que será este el modo en que recibiremos las imágenes del Frame Grabber.

El modelo RGB [ENAL2000] se representa mediante un sistema cartesiano cuyo subespacio es el cubo unidad (lado=1). Cada color es una punto determinado por un vector que tiene como origen el de coordenadas y acaba en dicho punto. El origen de coordenadas representa el negro (0,0,0).La diagonal principal representa los niveles de grises, y su extremo el blanco (1,1,1). Cada eje está asociado a un color (rojo, verde y azul).

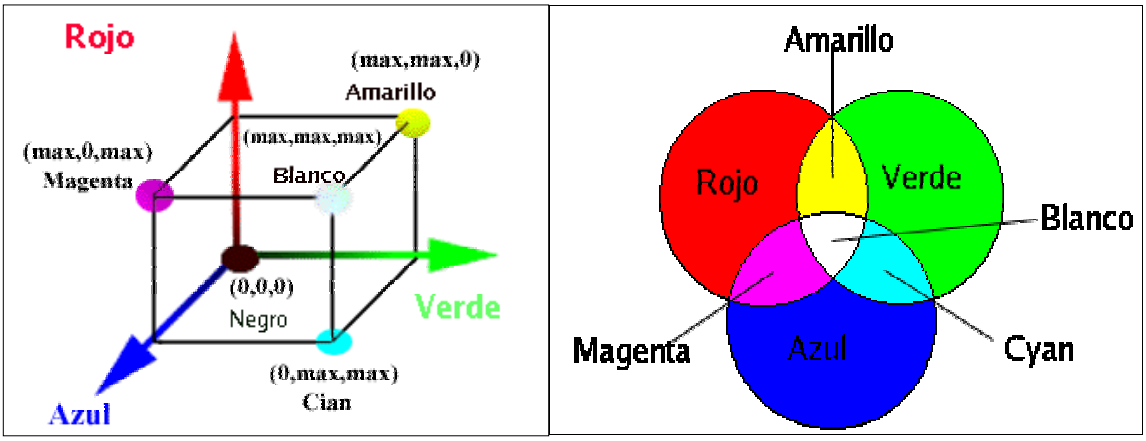


Fig 3.9 Modelo RGB

Para conseguir los colores (Fig. 3.9), se combinan los tres componentes básicos (colores primarios: rojo, verde y azul), al superponerse crean los colores secundarios: Cian, Magenta y Amarillo.

Color= ? Rojo + ? Verde +? Azul

Una variante es el **modelo RGBA**. Se trata del mismo modelo RGB, pero con otra propiedad: canal *alfa*. Este canal se usa como un índice de la transparencia en un píxel. Esto nos sirve a la hora de mezclar varios colores designados para un solo píxel. Este modelo es más reciente y se suele usar para crear efectos y técnicas visuales como "anti-aliasing", niebla, llamas, y objetos semi-transparentes: cristal, agua, vidrieras, etcétera.

Como veremos en el apartado 3.3.1, el modo RGBA es el que solicitaremos al tomar imágenes de la grabadora.

3.3 LIBRERÍA BTTV_MARTE:

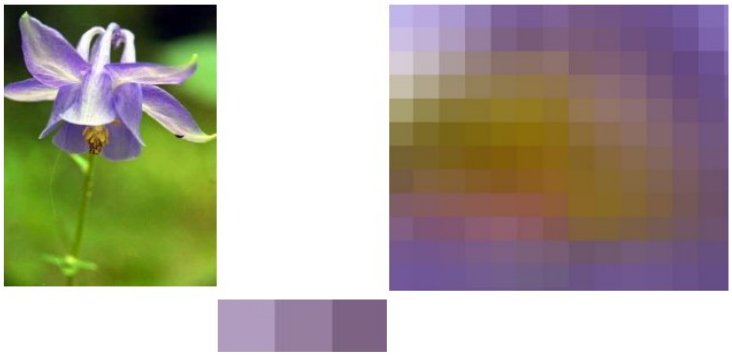
3.3.1 PARÁMETROS EN LA LIBRERÍA:

Accederemos a las imágenes captadas por la cámara a través de la librería BTTV_MaRTE desarrollada en la Universidad de Cantabria. Esta librería ofrece varias opciones a la hora de inicializar la cámara y cargar las imágenes, descritos a continuación:

- Podemos escoger la resolución de las imágenes, siempre y cuando se cumpla la relación de aspecto 4:3, por eso, aunque en principio la resolución máxima de la cámara es 768x576 pixeles, escogeremos una resolución de 640x480 pixeles para mantener la citada relación de aspecto.
- La librería bttv_marte permite escoger entre modo entrelazado y no entrelazado (ver apdo. 3.1.1.2). Tomamos la segunda para evitar la distorsión introducida por el efecto del movimiento de la bola, teniendo así imágenes cada 20 ms (en vez de cada 40 ms, obtenemos el campo par, y 20ms después el impar). La librería nos devuelve un solo campo, del que deshecha la mitad de líneas verticales para mantener la proporcionalidad de la imagen, es decir, nos devuelve imágenes de una resolución de 320 x 240 pixeles.
- Podemos utilizar varios buffers de datos, para trabajar con uno sin que la nueva captura lo sobreescriba. Como terminaremos las operaciones antes de la nueva captura, sólo necesitaremos sólo uno.
- Entre los modos en color, escogeremos una de las profundidades de píxel que ofrecen mayor calidad, es decir, RGB_24 ó RGB_32 (ver apdo. 3.2.3.1). La opción RGB_24, almacena 3 bytes por cada píxel, con los contenidos de color de cada uno de ellos. La opción RGB_32, almacena 4 bytes por píxel,que es precisamente el tamaño con el que se guarda una variable de tipo entero, 3 de ellos con contenido de color, y un cuarto byte conocido como byte Alfa (ver apdo. 3.2.3.1). De esta forma, cada vez que pedimos un entero de memoria, se nos devuelve un píxel completo, mientras que con la opción RGB_24 obtendríamos parte del píxel siguiente o anterior.

La primera opción utiliza menos memoria, sin embargo la segunda es más eficiente, ya que los pixeles pueden ser copiados fácilmente de unas posiciones de memoria a otras con sólo referirse a ellos como variables de tipo entero. Conocida la posición de memoria en la que está almacenado el buffer, podemos acceder inmediatamente a la posición de cualquier dato sin más que conocer la resolución utilizada.

La figura 3.10 muestra una comparación entre la forma de almacenar datos en memoria de los modos RGB y RGBA:



R:176	G:156	B:191	R:150	$FF00	R:176	G:156	B:191	α
G:126	B:162	R:125	G:98	$FF04	R:150	G:126	B:162	α
B:133				$FF08	R:125	G:98	B:133	α
				$FF0C				

Fig 3.10 Comparación modos RGB y RGBA

3.3.2 INTERFAZ CON BTTV_MARTE:

El tipo de dato que utiliza la librería para almacenar el buffer que contiene la imagen, es el siguiente:

- *struct **video_multiple_buffer**{*

```
unsigned char        frame_grabber;******
unsigned char        n_buffers;      /* Número de buffers */
void                 *base[NUMBER_BUFFERS];
unsigned short int   height,width;   /* Resolución de la imagen*/
unsigned char        depth;          /* Profundidad de píxel*/
unsigned short int   bytesperline;
unsigned char        format;         /* Formato de la imagen*/
unsigned char        mode;           /* Modo: entrelazado o no entrelazado*/
struct timespec      timestamp_odd[NUMBER_BUFFERS];   /*Instante de captura*/
struct timespec      timestamp_even[NUMBER_BUFFERS];  / *de cada campo*/
};
```

En *nbuffer*, indicamos, de los posibles buffers, el que queremos cargar.

En *base*, tenemos un puntero a la posición de memoria del buffer, a continuación, el tamaño de la imagen (en *height y width*) y en *depth,* la profundidad de píxel. *Bytesperline* almacena el

número de bytes que ocupa cada línea de la imagen, debe equivaler por tanto al producto de *depth* y *width*. Los últimos datos, *timestamp_odd y timestamp_even*, contienen el instante de tiempo en el que fue tomado cada campo de la imagen (par e impar).

Para definir el formato de la imágen, bttv_MaRTE define los siguientes tipos:

- struct **video_picture**

```
{
#define VIDEO_PALETTE_GREY      1     /* Limagen en blanco y negro */
#define VIDEO_PALETTE_HI240     2
#define VIDEO_PALETTE_RGB565    3     /* 565 16 bit RGB */
#define VIDEO_PALETTE_RGB24     4     /* RGB  en 24 bits*/
#define VIDEO_PALETTE_RGB32     5     /* RGB  en 32 bits*/
#define VIDEO_PALETTE_RGB555    6     /* 555 15bit RGB */
#define VIDEO_PALETTE_YUV422    7     /* YUV422 */
#define VIDEO_PALETTE_YUYV      8
#define VIDEO_PALETTE_UYVY      9
#define VIDEO_PALETTE_YUV420    10
#define VIDEO_PALETTE_YUV411    11    /* YUV411 */
#define VIDEO_PALETTE_RAW       12    /* RAW captura (BT848) */
#define VIDEO_PALETTE_YUV422P   13    /* YUV 4:2:2 Planar */
#define VIDEO_PALETTE_YUV411P   14    /* YUV 4:1:1 Planar */
#define VIDEO_PALETTE_YUV420P   15    /* YUV 4:2:0 Planar */
#define VIDEO_PALETTE_YUV410P   16    /* YUV 4:1:0 Planar */
#define VIDEO_PALETTE_PLANAR    13
#define VIDEO_PALETTE_COMPONENT 7
};
```

Escogeremos el modo Video_Palete_RGB32, que corresponde al RGBA descrito en el apartado anterior.

Para indicar modo entrelazado o no entrelazado, *format* se define:

#define INTERLACED 0

#define NON_INTERLACED 1

Con todo lo anterior, utilizamos las siguientes funciones de la librería bttv_marte:

- **Inicialización del buffer de datos**, con *height*=240, *width*=320. *format=video_palette_rgb _* 32 y *mode =non_interlaced*

```
int init_video_multibuffer (unsigned int                height,
                            unsigned int                width,
                            unsigned short int          format,
                            unsigned char               mode,
                            struct video_multiple_buffer *buffer);
```

- **Inicialización de la grabadora**:

```
extern int start_frame_grabber (unsigned int                frame_grabber,
                                struct video_multiple_buffer *fbuf);
```

Frame_grabber es el identificador del buffer.

- **Obtener imagen nueva**: La librería bttv_marte permite cargar la última imagen recogida por la cámara que en el caso peor tendría un periodo de antigüedad (40 ó 20 ms en función de si trabajamos en modo entrelazado o no), o bien esperar y obtener la siguiente imagen. Utilizaremos la segunda opción, es preferible esperar una imagen válida que procesar imágenes de las que desconocemos su validez. Para ello utilizamos la función:

```
extern int wait_for_next_image ( struct video_multiple_buffer *fbuf,
                                 void                         *imagen,
                                 size_t                       size_of_image,
                                 struct timespec              *timestamp);
```

Con *size_of_image = width*height*4* (resolución * bytes de profundidad de cada píxel). En **fbuf*, indicamos el número que identifica al buffer con el que estamos trabajando y en **imagen* obtenemos el último campo capturado.

Esta función devuelve en *timestamp* además el instante de tiempo en el que se tomó la imagen. Es de carácter bloqueante, es decir, no devuelve el control hasta ofrecer una imagen válida, sin embargo, como veremos en el apartado 5.5.3, el procesado de imagen se sincroniza con la captura a partir de la segunda iteración del ciclo, por lo que este bloqueo no supondrá ningún problema.

- Dibujar imágenes por pantalla:

  *void **elaborate_image** (unsigned char *imageptr,*

  struct timespec time)

Esta función dibuja en pantalla la imágen que se encuentra en la posición de memoria indicada por **imageptr*, nos será muy útil en las etapas de calibración de la librería, ya que en MARTE no podemos acceder a los resultados a través de ficheros. En *time*, indicamos el instante de la captura.

3.4 LIBRERÍA GANDALF:

Gandalf [2] es una librería C diseñada para el desarrollo de aplicaciones de visión por ordenador. Fue desarrollada por *Imagineer Software Ltd* para desarrollar "MoKey", un software de edición de vídeo. Gandalf contiene cuatro secciones: 1) Paquete común de estructuras simples y rutinas usadas por otros paquetes; 2) Un paquete de Algebra Lineal con un amplio número de rutinas para manipulaciones matriciales y vectoriales; 3) un paquete para imágenes, definiendo una estructura de imagen de propósito general y rutinas de manipulación de imágenes a bajo nivel; 4) Un paquete de visión por ordenador, conteniendo un número de rutinas de procesamiento de imágenes.

El paquete imagen contiene funciones de creación y manipulación de imagen, soporta nivel de grises y RGB con y sin canal Alfa (ver apdo 3.2.3.1) con varias profundidades de píxel.

Esta librería, nos servirá en Linux para realizar pruebas, permitiéndonos acceder a la información contenida en una imagen guardado como archivo, a través de nuestras funciones en C. Para la versión MaRTE de la librería, sustituiremos Gandalf por la librería Bttv_Marte.

Gandalf distingue entre el formato de imágenes externas y el formato que utiliza internamente para representar las imágenes. Soporta seis tipos de ficheros de imagen externa: PBM (Portable Bit Map), PGM (Portable Grey-level Map), PPM (Portable píxel Map), PNG (Portable Network Format), TIF (Tag Image File Format) y JPEG (Joint Photographic Experts Group). PNG, TIF y JPG, introducen compresión, son más complejos y requieren la instalación de librerías adicionales. Por este motivo, analizaremos imágenes de tipo PPM.

Los tres primeros tipos, PBM, PGM y PPM, pertenecen al grupo de ficheros PNM, que significa *Portable aNy Map,* pensado para cubrir las 6 variaciones de los bit/gray/píxel maps. El archivo en el que se guardan, es simplemente un fichero de texto en el que se indican el número mágico, el número de pixeles de ancho de la imagen, el número de pixeles de alto de la imagen, el número de colores representado y, por último, el valor de cada byte de la imagen. El número mágico, indica el formato del fichero, con las siguientes posibilidades:

- P1, si el fichero es ASCII y en Blanco/Negro.

- P2, si el fichero es ASCII y en escala de grises.
- P3, si el fichero es ASCII y en color.
- P4, si el es binario y en Blanco/Negro
- P5, si el es binario y en escala de grises.
- P6, si el es binario y en color.

En versión ASCII, los archivos .ppm son grandes e ineficientes, pero son fáciles de usar y de inspeccionar a simple vista. En versión binaria los archivos son mucho mas pequeños, y se escriben más rápido. Gandalf admite el tipo binario, con lo cual, las imágenes que vamos a utilizar, serán de tipo PPM P6.

3.4.1 INTERFAZ CON LA LIBRERÍA GANDALF

Los tipos de datos utilizados en la interfaz con la librería:

- Los formatos de imagen internos disponibles en la librería se definen en el tipo **Gan.ImageFormat**:

```
typedef enum {
        GAN_GREY_LEVEL_IMAGE,          /* Imagen en escala de grises*/
        GAN_GREY_LEVEL_ALPHA_IMAGE,    /* Imagen en escala de grises con
                                          canal alfa */
        GAN_RGB_COLOUR_IMAGE,          /* Imagenes RGB en color */
        GAN_RGB_COLOUR_ALPHA_IMAGE,    /* Imagenes RGB en color con
                                          canal alfa* /
        GAN_VECTOR_FIELD_2D,           /*Imagenes 2D con vectores 2D */
        GAN_VECTOR_FIELD_3D }          /*Imagenes 3D con vectores 3D */
 Gan_ImageFormat;
```

- Los tipos de datos para las imágenes externas vienen definidos en Imagen_IO.h:

```
typedef enum{
  GAN_PNG_FORMAT,
  GAN_PBM_FORMAT,
  GAN_PGM_FORMAT,
  GAN_PPM_FORMAT,
  GAN_TIFF_FORMAT,
  GAN_JPEG_FORMAT,
  GAN_UNKNOWN_FORMAT
} Gan_ImageFileFormat;
```

- Para referirse a las imágenes internas, Gandalf utiliza un puntero a imagen, es un tipo de dato interno y se definen funciones para operar con él:

  *Gan_Image * pImage;*

- Existen distintos tipos para describir pixeles, en función de si la imagen es en blanco y negro, color, lleva canal alfa... Utilizaremos el tipo Gan_RGBpíxel_uc, que se refiere a imágenes RGB, sin canal alfa, y guarda, por cada píxel tres unsigned characters, que ocupan un Byte cada uno:

```
typedef struct Gan_RGBpíxel_uc {
        unsigned char R,
        unsigned char G,
        unsigned char B,
} Gan_RGBpíxel_uc;
```

Funciones utilizadas:

- Reservar espacio de memoria para almacenar una imagen tipo RGB (el tamaño de la imágen en pixeles se indica a través de *row* y *column*)y liberarlo:

```
pImage = gan_image_alloc_rgb_uc (int row,
                                 int column);
gan_image_free (GanImage    *pImage);
```

- Leer una imagen desde un archivo PPM 6:

```
Gan_Image *gan_image_read ( const char          *filename,
                            Gan_ImageFileFormat file_format,
                            Gan_Image           *image );
```

En nuesto caso, *file_format* será *GAN_PPM_FORMAT.*

- Escribir una imagen en un archivo PPM 6:

```
Gan_Bool  gan_image_write ( const char          *filename,
                            Gan_ImageFileFormat file_format,
                            Gan_Image           *pImage );
```

- Leer un píxel de una imagen:

```
Gan_píxel  gan_image_get_pix (Gan_Image      *pImage
                              int            row
                              int            column);
```

- Modificar un píxel en una imágen RGB:

```
Gan_image_set_pix_rgb_uc (Gan_Image          *pImage,
                          int                row,
                          int                column,
                          Gan_RGBpíxel_uc    * rgbucpíxel);
```

4. CALIBRADO DE LA CÁMARA:

4.1 DISTORSIÓN RADIAL:

El fenómeno que queremos corregir es el de distorsión radial, es decir, la distorsión producida por la cámara de las lineas rectas de la imagen real, que aparecen como lineas curvas. Esta distorsión, provocada por efectos en la curvatura de las lentes, hace que la información útil que vamos a extraer de la imagen no sea todo lo exacta y real que queremos, como se observa en la figura 4.1:

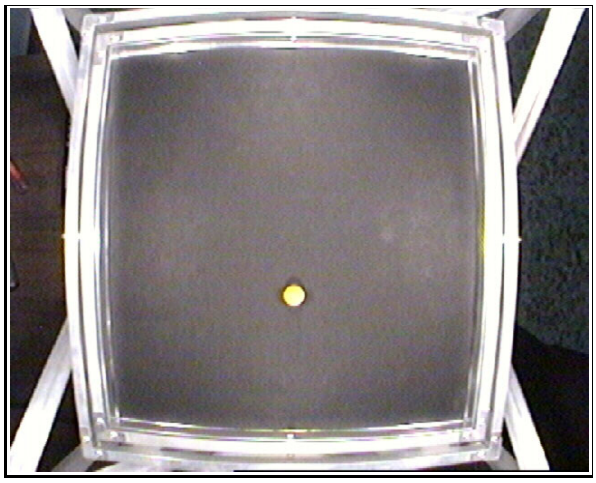


Fig 4.1 Distorsión radial

Si bien existen cámaras con lentes que presentan una distorsión radial mínima y una distancia focal muy elevada, son muy costosas para una aplicación como la nuestra.

Existen modelos como el de **Tsai** [TSA1987] para calibrar una cámara y corregir su distorsión radial, sin embargo, requiere conocer algunas características físicas que en nuestro caso no están disponibles y cuyo calculo se escaparía del objetivo de este proyecto. Por esta razón, se necesitan modelos de distorsión y métodos de corrección de imágenes que no requieran información a priori de la cámara, sino que obtengan la información necesaria únicamente a partir de la imagen distorsionada.

4.4.1 MODELO DE DISTORSIÓN RADIAL:

Existen métodos que utilizan información a priori de la escena real, como por ejemplo el conocimiento de ciertas propiedades de los objetos capturados, como linealidad, paralelismo, simetría.

Utilizando una rejilla de calibración como la de la figura 4.2, se puede obtener el modelo de distorsión y la rectificación de la imagen de una forma relativamente rápida, tomando 3 o 4 lineas de referencia, al aprovechar las propiedades conocidas de la imagen, que sabemos que corresponde a una rejilla de cuadros paralelos [VOSS2001]:

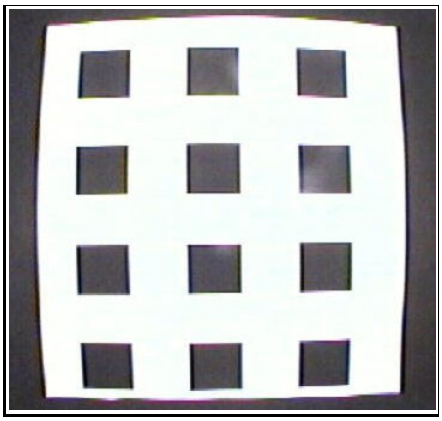
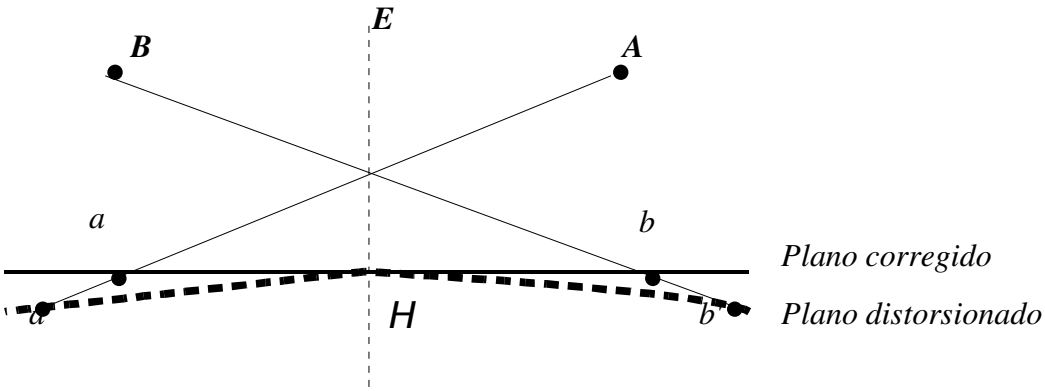


Fig 4.2 Rejilla de calibración

Para modelar la distorsión radial, supondremos que la distorsión es del tipo no lineal, que tiene solo una parte radial y que el centro de la imagen coincide con el centro de distorsión.

Llamaremos coordenadas “distorsionadas” (a', b') a las coordenadas de la imagen distorsionada y coordenadas “corregidas” (a,b) a las coordenadas de la imagen corregida. Las coordenadas que obtenemos a partir de las imágenes que nos da la cámara, son coordenadas distorsionadas, que pueden coincidir o no con las corregidas, como se aprecia en la figura 4.3.

Un punto espacial A se proyecta en el plano distorsionado como *a'*, la distorsión radial se aprecia en que a este punto le correspondería la posición *a.*



4.3 Planos distorsionado y corregido

Como se aprecia, el centro de distorsión *H* coincide con el centro de la imagen y en ese punto las coordenadas distorsionadas coinciden con las corregidas. Además, se aprecia en la figura que la distorsión radial es simétrica con respecto del eje E, ya que dos puntos simétricos con respecto del centro H se desplazan en la misma medida.

Puede apreciarse mejor en una imagen real distorsionada Fig 4.3 , se observa la simetría rotacional con respecto al eje óptico; Las esquinas señaladas en la imagen están distorsionadas, sus puntos corregidos están sobre la linea que pasa por el punto distorsionado y el centro de distorsión *H:*

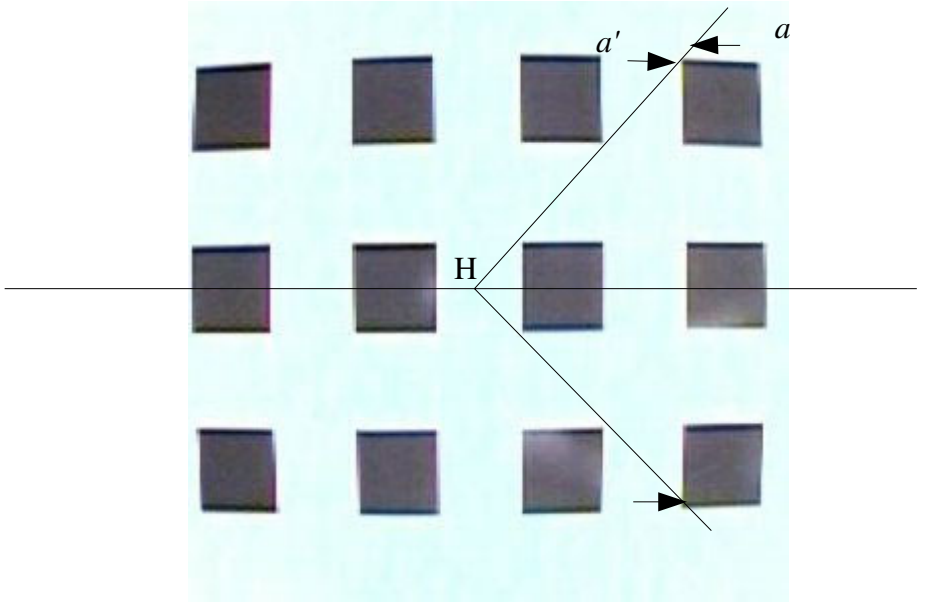
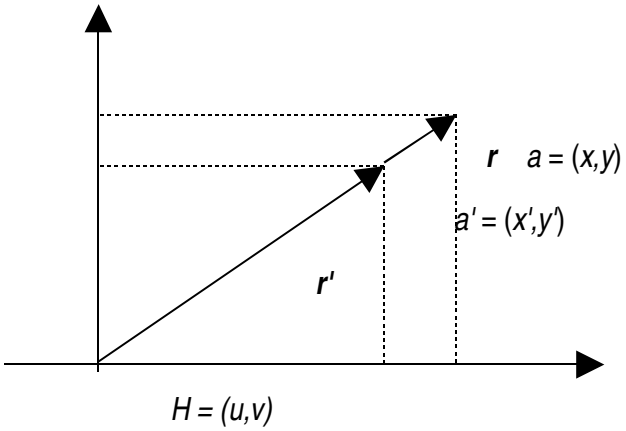


Fig 4.3.Simetría de la distorsión

Para efectuar la corrección, haremos una proyección central y una transformación no lineal. La proyección central se llevara a cabo mediante un cambio de coordenadas, estableciendo como origen el centro de la imagen. Necesitamos a continuación una transformación no lineal, ya que los puntos próximos al centro, serán desplazados en distinta medida que los de los extremos, esta transformación se puede escribir a partir de las expresiones 4.1 propuestas en [VOSS2005].

Podemos ver en la figura que los puntos *a* y *a'* están sobre la misma recta, tenemos por lo tanto una relaciona entre las distancias *r* y *r'* (distancias de los puntos corregido y distorsionado al centro de la imagen):



Si llamamos:

(x,y) al punto corregido

(x',y') al punto distorsionado

$$r=\sqrt{(x-u)^2+(y-v)^2} \qquad r'=\sqrt{(x'-u)^2+(y'-v)^2}$$

$$\frac{x-u}{x'-u}=\frac{r}{r}, \qquad \frac{y-v}{y'-v}=\frac{r}{r},$$

$$r=\frac{r'}{[1-f(r')]}$$

$$r'=r[1-f(r')] \qquad \text{(Exp 4.1)}$$

La expresión (Exp. 4.1) relaciona las distancias de los puntos distorsionado y corregido respecto del centro, como vemos en la figura XX, la distancia entre los radios r y r' aumenta al alejarse del centro de la imagen y es casi nula sobre este.

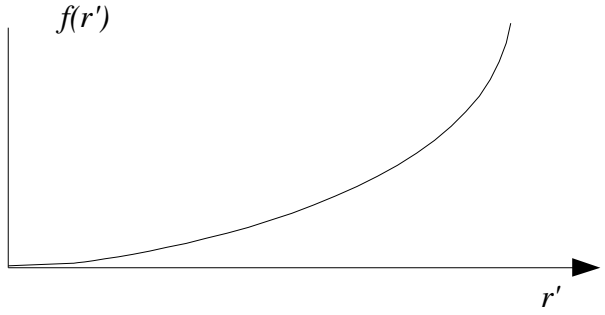
El rango de variación de f(r') varia de (0,1):

- Cerca del centro hay poca distorsión, por lo que $f(r')\to 0$ y el factor $[1-f(r')]\to 1$, con lo que r'=r.
- Al alejarse del centro, $f(r')$ aumenta, el factor $[1-f(r')]$ disminuye, y $r\neq r'$

El problema del modelado de la distorsión es encontrar la función f(r') para una cámara dada. Diversos trabajos publicados como Tsai y Stein sobre la calibración de distorsión radial, llegan a una buena aproximación de esta función:

$$r'=r(1+d_2\cdot r'^2+d_4\cdot r'^4)$$

Exp. 4.2



Siendo d2 y d4 los coeficientes
pares de aproximación de r', los impares se desprecian debido a la simetría radial de distorsión.

4.2 PARÁMETROS DE LA DISTORSIÓN RADIAL: MÉTODO BRAÜER

Braüer [BRBU2000] propuso un método sencillo para la corrección de la distorsión radial basándose en en la ecuación Exp. 2 . Podemos despejar de esta:

$$r=\frac{r}{1+d_2\cdot r'^2+d_4\cdot'^4} \qquad \text{Exp. 4.3}$$

Sean *P=(X,Y),* las coordenadas del centro de distorsión (el centro de la imagen), a'=(x',y') las de un punto distorsionado de la imagen, a=(x,y) las de un punto corregido y r' y r las distancias de los puntos distorsionado y corregido al centro de la imagen. Entonces podemos escribirlas coordenadas de un punto corregido a partir de 4.2:

$$x=X+\frac{x'-X}{1+d_2\cdot r'^2+d_4\cdot r'^4} \qquad y=Y+\frac{y'-Y}{1+d_2\cdot r'^2+d_4\cdot r'^4} \qquad \text{Exp. 4.4}$$

Y la distancia al cuadrado del punto distorsionado al centro:

$$r'^2=(x'-X)^2+(y'-Y)^2 \qquad \text{Exp. 4.5}$$

Ahora trasladamos el origen de la imagen al centro de la la imagen, esto es: X=0,Y=0, luego podemos reescribir:

$$x'=\frac{r'}{r} \qquad y'=\frac{r'}{r} \qquad \frac{r'}{r}=1+d_2\cdot r'^2+d_4\cdot r'^4 \qquad \text{Exp 4.6}$$

Si tomamos tripletas de puntos de la misma linea distorsionada (curva definida por p_1',p_2',p_3',) como muestra la figura 4.4:

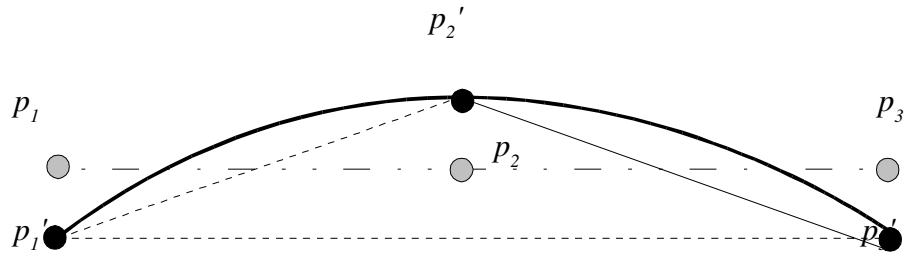


Fig. 4.4 Tripletas de puntos

Los puntos distorsionados forman un triángulo A_{123}, cuyo área debe ser cero, ya que corresponden a una linea. Podemos despejar los parámetros d2 y d4 de la ecuación 4.3 minimizando este área:

$$A_{123}=\frac{1}{2}\cdot\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}=0$$

en el caso ideal, sin distorsión, $d_2=d_4=0$, luego utilizando las ecuaciones 4.6:

$$A_{123}=\frac{1}{2}\cdot\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}=\frac{1}{2}\cdot\begin{vmatrix} x_1' & y_1' & \frac{r_1'}{r_1} \\ x_2' & y_2' & \frac{r_2'}{r_2} \\ x_3' & y_3' & \frac{r_3'}{r_3} \end{vmatrix}\cdot\frac{1}{\frac{r_1'}{r_1}\frac{r_2'}{r_2}\frac{r_3'}{r_3}}$$

dado que $\frac{1}{\frac{r_1'}{r_1}\frac{r_2'}{r_2}\frac{r_3'}{r_3}}\neq 0$ se llega a que: $\begin{vmatrix} x'_1 & y'_1 & 1+d_2r'^2+d_4r'^4 \\ x'_2 & y'_2 & 1+d_2r'^2+d_4r'^4 \\ x'_3 & y'_3 & 1+d_2r'^2+d_4r'^4 \end{vmatrix}=0$

Resolviendo el determinante y agrupando términos se llega a:

$$\delta_{123}= d_4[r_1'^4 s_1+r_2'^4 s_2+r_3'^4 s_3]+d_2[r_1'^2 s_1+r_2'^2 s_2+r_3'^2 s_3]+s_1+s_2+s_3 \quad \text{Exp.4.7}$$

$$s_1=x_2'y_3'-x_3'y_2'$$

$$s_2=x_3'y_1'-x_1'y_3'$$

$$s_3=x_1'y_2'-x_2'y_1'$$

Para cada tripleta de puntos p_1',p_2',p_3', se construye una ecuación $\delta^{(i)}_{123}$, y con todas ellas un sistema de ecuaciones lineales sobredeterminado don dos incógnitas; d_2 y d_4. Se llega a un problema de optimización:

$$\sum_{i=1}^{z}[\delta_{123}^{(i)}]^2=f(d_2,d_4)\rightarrow min$$

para z tripletas de puntos. La solución se encuentra por optimización de mínimos cuadrados tomando suficientes tripletas.

4.3 IMPLEMENTACIÓN DEL ALGORITMO DE CORRECCIÓN:

El proceso de corrección se lleva a cabo por un proceso iterativo. Al empezar asignamos el centro de la imagen como el punto principal ($p^{[0]}$), se aplican las coordenadas de las tripletas de puntos a la optimización (XX) para calcular los primeros valores de los coeficientes $d_2^{[1]}$ y $d_4^{[1]}$. A continuación se actualiza el valor de las coordenadas del punto principal y suponiendo correctos d_2 y d_4 hallaremos la medida del error M_T para un grupo de puntos al rededor de $P^{[0]}$. El punto que

tenga la medida de error M_T menor, será el nuevo punto principal $P^{[i+1]}$ de la imagen para la nueva iteración. Se repite el proceso y se recalculan $d_2^{[2]}$ y $d_4^{[2]}$. El proceso termina cuando el valor del coeficiente de error ya no decrece o cuando no varían los coeficientes.

Utilizamos una rejilla como la mostrada en la figura y extraemos de ella las esquinas de los cuadrados de la imagen, pensada para facilitar este proceso. Elegiremos lineas verticales con seis puntos cada una:

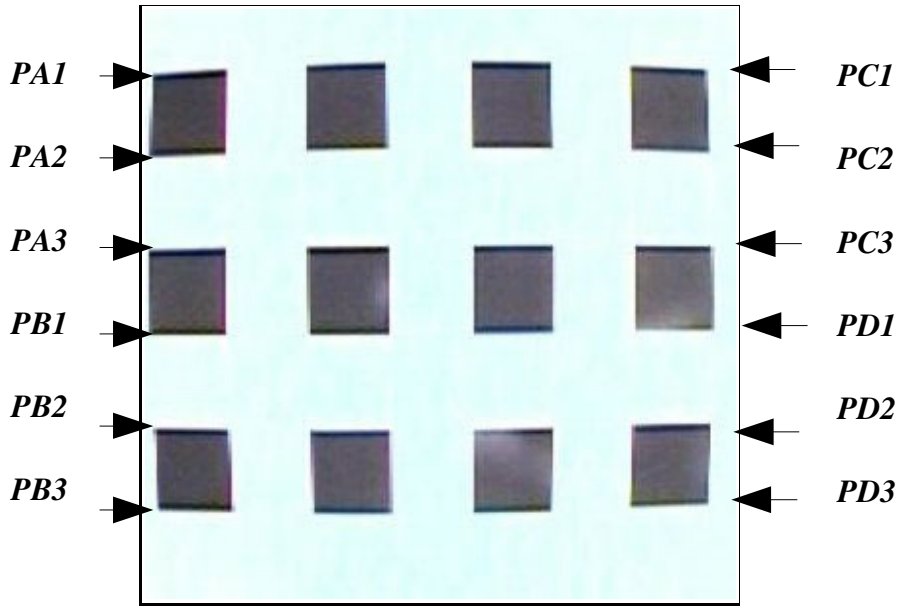


Fig. 4.5 Tripletas escogidas

De este modo tenemos cuatro tripletas de puntos. En este punto debemos llegar a un compromiso entre el número de tripletas escogidas y el coste computacional que esto requiere.

Con las coordenadas de cada tripleta de puntos se calculan los coeficientes de la ecuación 4.7 construyendo un sistema de ecuaciones lineales sobredeterminado:

$$\delta_{123}^{[i]} = d_4 A_i + d_2 B_i + C_i$$

Habiendo llamado, para simplificar:

$$A_i = r_1'^{4[i]} s_1^{[i]} + r_2'^{4[i]} s_2^{[i]} + r_3'^{4[i]} s_3^{[i]}$$

$$B_i = r_1'^{2[i]} s_1^{[i]} + r_2'^{2[i]} s_2^{[i]} + r_3'^{2[i]} s_3^{[i]}$$

$$C_i = s_1^{[i]} + s_2^{[i]} + s_3^{[i]}$$

Este sistema podemos resolverlo por medio de MATLAB™, que ofrece funciones para la solución de este tipo de sistemas lineales.

Las coordenadas de los puntos escogidos respecto del centro de la imagen que tomamos para la calibración son:

	Linea A	***Linea B***	***Linea C***	***Linea D***
Punto 1	(-146,-117)	*(-148,-16)*	***(135,-119)***	(138,13)
Punto 2	(-148,-74)	(-145,-64)	(136,-78)	(136,62)
Punto 3	(-149,-27)	(144,-104)	(138,-30)	(135,102)

Los valores de los parámetros que obtenemos son d_2=-1.106 · 10^{-6} y d_4=3.128 · $10^{-12.}$

Con estos valores, aplicaremos directamente las expresiones 2.4 para obtener el valor del punto corregido. Sin embargo, en otros tipos de aplicaciones se desea reconstruir la imagen completa, en ese caso, sería necesario interpolar para recuperar los niveles de gris (o RGB) y evitar así los efectos de la digitalización.

5. PROCESADO DE IMAGEN:

El proceso de búsqueda del centro de la bola se lleva a cabo en varios pasos. Partimos de una imagen RGBα, es decir, tenemos tres campos de información (el canal alfa no lo procesaremos), contenido en rojo, verde y en azul, cada uno de ellos con una profundidad de un byte, lo que significa que tenemos 256 posibles cantidades de los tres colores. El principal problema de este tipo de imágenes es que todos los puntos de la imagen contendrán los tres colores, en mayor o menor proporción, es por ello que necesitamos una etapa de segmentación, en la que distingamos los puntos que pertenecen a la bola de los que no. A continuación, aplicaremos una erosión para eliminar los puntos aislados exteriores a la bola, para finalmente calcular el centro.

A este proceso, se le conoce como procesado de imagen [REK1994]:

Se entiende por procesamiento en el dominio del espacio, la realización de operaciones directamente sobre el valor de los pixeles que forman la imagen de entrada. En esta categoría quedan comprendidos los siguientes tipos de procesamiento: Operaciones lógico-aritméticas, transformaciones punto a punto, transformaciones geométricas y convolución bi-dimensional.

5.1 ETAPAS FUNDAMENTALES DEL PROCESADO DE IMÁGENES:

Los pasos principales que tiene que seguir un sistema de captación de imágenes para obtener la imagen y luego poder interpretarla se resumen en el siguiente esquema [DELA2001]:

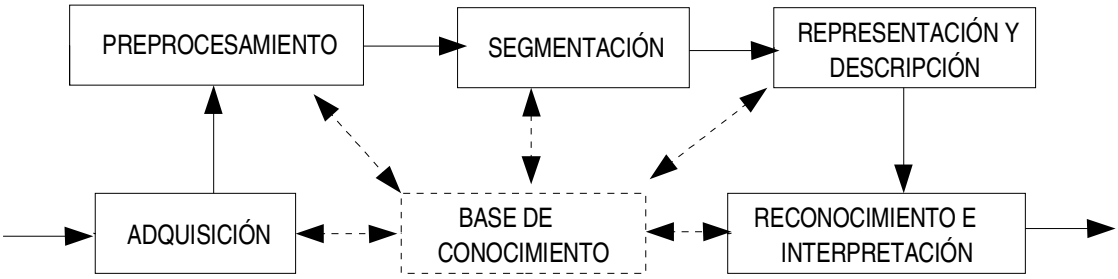


Fig 5.1 Etapas del procesado de imagen

Una vez que se ha obtenido la imagen digital (adquisición), la siguiente etapa trata del ***preprocesamiento*** de esa imagen [DELA2001]. La función básica del preprocesamiento es la de mejorar la imagen de forma que se aumenten las posibilidades de éxito en los procesos posteriores. Un preprocesamiento típico consiste en utilizar técnicas para mejorar el contraste o eliminar ruidos.

La siguiente etapa es la ***segmentación***. Definida de una forma general, la segmentación consiste en partir una imagen de entrada en sus partes constituyentes u objetos. En cualquier imagen se encontrarán presentes uno o varios objetos localizados en un entorno, el objetivo de la segmentación es separar dichos objetos del medio en que se encuentran y distinguirlos entre sí. Se basa en tres propiedades:

1. Similitud: Cada uno de los pixeles de un elemento tiene valores parecidos para alguna propiedad
2. Discontinuidad: Los objetos destacan del entorno y tienen por tanto unos bordes definidos.
3. Conectividad: los pixeles pertenecientes al mismo objeto tienen que ser contiguos, es decir, estar agrupados.

Después del proceso de segmentación se tienen tres tipos de regiones: objetos, fondo y agujeros (puntos que no pertenecen al objeto pero están rodeados por él).

En general, la segmentación autónoma es una de las labores más difíciles del tratamiento digital de imágenes. Por una parte, un procedimiento de segmentación demasiado tosco dilata la solución satisfactoria de un problema de procesamiento de imágenes. Por otra, un algoritmo de segmentación débil o errático casi siempre garantiza que tarde o temprano habrá un fallo.

A la salida del proceso de segmentación, habitualmente tendremos o bien el contorno de una región o bien todos los puntos de una región determinada.

Una vez terminados los algoritmos de segmentación, cada objeto presente en la imagen puede ser analizado de forma individualizada. Pero primero, habrá que resolver problemas heredados de la segmentación, como pixeles mal clasificados.

La etapa de ***presentación***, es la que estructura los datos devueltos (por ejemplo en nuestra aplicación podemos devolver un array conteniendo puntos de la bola, o el centro de la misma, como es nuestro caso).

La etapa de ***reconocimiento*** tiene como objetivo identificar el objeto representado en la imagen. Por ejemplo, reconstruir una figura a partir del array de puntos devuelto en la etapa de presentación. En nuestro caso, las últimas dos etapas no se diferencian, ya que, como resultado, devolveremos un único punto, el centro.

Todas las etapas pueden tener relación con la ***base de conocimiento***, su misión es interrelacionarse con todas las etapas, permitiendo ir de una etapa a la anterior o a la siguiente, y almacenar información.

5.2 FUNDAMENTOS TEÓRICOS

5.2.1 OPERACIONES PUNTO A PUNTO

Una vez que las imágenes han sido digitalizadas y guardadas en forma de matriz en el correspondiente sector de memoria (*buffer*) del computador, resulta inmediata la posibilidad de realizar operaciones aritméticas y lógicas entre ellas. Estas operaciones son realizadas punto a punto entre pixeles correspondientes.

Este tipo de procesamiento consiste fundamentalmente en la reasignación de niveles de color (o niveles de gris, según la aplicación) punto a punto, a través de una función de transformación de la forma s = T (r) . Esta función se encuentra representada en la figura 5.2. En esta función, los niveles de gris de la imagen de entrada están representados por la variable r, mientras que la variable s representa los niveles de gris de la imagen de salida. Imax es el rango máximo del píxel.

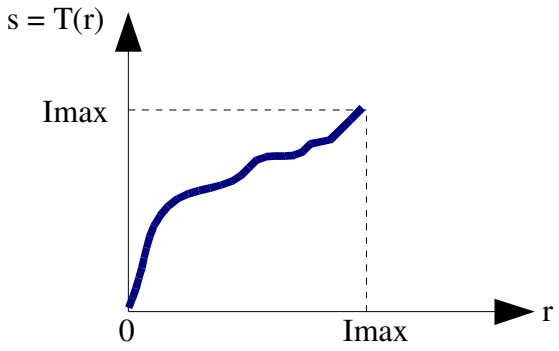


Fig 5.2 Función de transformación de niveles de gris punto a punto.

5.2.1.1 UMBRALIZACIÓN:

Una de las transformaciones punto a punto más común es la ***umbralización***, común en la etapa de segmentación de imágenes.

La umbralización consiste en el paso de una imagen cromática o monocromática (escala de grises), a una imagen binaria. En esta última aparecerán sólo dos valores de intensidad de luz en función de los valores previos de los pixeles en la imagen original. Se intenta que los pixeles correspondientes a objeto tengan el mismo valor y este sea diferente al de los pixeles que corresponden a fondo. Para conseguirlo, debe buscarse un intervalo de valores de color dentro del

cual los pixeles se consideren perteneciente a objeto. Luego, para la determinación de los objetos basta localizar grupos de pixeles de igual valor.

A menudo se utilizan indistintamente los términos umbralización y binarización. En sentido estricto, umbralizar implica distinguir los pixeles que superan un umbral de los que no. Si se utilizan únicamente dos valores para representarlos, entonces hablamos de binarización.

En las imágenes binarias, uno de los valores, corresponde a regiones externas (fondo), mientras que el resto de pixeles corresponden a objetos de la imagen. Esta asignación se conoce como ***polaridad***, que normalmente corresponde a blanco y negro. Numéricamente, los dos valores son comúnmente 0 para el negro y 1 o 255 para el blanco. Es decir, una vez terminado el proceso de binarización, obtenemos una imagen en la que los puntos que representan objetos de interés tienen valor 1 ó 255, y el resto de puntos, de valor 0, representan el fondo.

La figura 5.3 ilustra este proceso. De los 255 posibles valores de la imagen original, solo existen dos posibles de salida, mínimo (negro) y máximo (blanco). Las zonas negras corresponden a los puntos de la imagen que no superan cierto umbral *u,*el resto, aparecen como blanco.

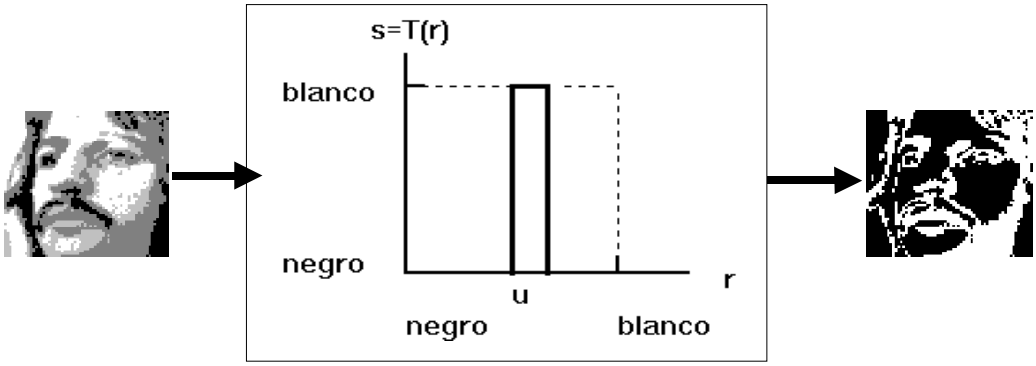


Fig 5.3: Ejemplo de binarización

5.2.1.2 HISTOGRAMA DE UNA IMAGEN:

Para la elección del umbral, es útil contar con el histograma de la imagen. El histograma contiene el número de pixeles que tienen el mismo valor de gris [DELA2001] (en imágenes RGB, tendríamos tres histogramas; uno por cada color). Pueden entenderse como la probabilidad de que un valor de gris (rojo, verde o azul) determinado aparezca en la imagen. Tiene la gran ventaja que condensa la información de la imagen en muchos menos valores y el gran inconveniente de que por sí solo no aporta ninguna información sobre la localización espacial de los pixeles. Así se tendrán infinidad de imágenes con el mismo histograma.

Sin embargo, en casos como el nuestro, podemos obtener un histograma separado conteniendo información únicamente de los puntos de la bola y compara éste con el de la imagen para fijar los umbrales.

5.2.2 OPERACIONES MORFOLÓGICAS:

La morfología matemática [PETM1996](tratado de formas) es una herramienta para el procesamiento y análisis de imágenes digitales, que ha sido desarrollada y explorada en las dos últimas décadas.

Su diferencia fundamental con los filtros lineales radica en el hecho de que el filtrado no se realiza a través de la discriminación del contenido armónico espectral de la imagen, sino directamente en el dominio del espacio y en función de su *forma*, es decir, la transformación ya no se realiza punto a punto como en el caso anterior, si no que el resultado depende de la relación de un píxel con los adyacentes. Las operaciones morfológicas simplifican imágenes y conservan las principales característica de forma de los objetos.

La morfología matemática se puede usar, entre otros, con los siguientes objetivos:

- Pre-procesamiento de imágenes (supresión de ruidos, simplificación de formas).
- Destacar la estructura de los objetos (extraer el esqueleto, detección de objetos, envolvente convexa, ampliación, reducción,...) y eliminar ruido producido en el proceso de segmentación.
- Descripción de objetos (área, perímetro,...)

Los operadores básicos de la morfología matemática son la *erosión* y la *dilatación*. ambos operadores toman dos datos de entrada; una imagen y un elemento estructural (kernel).

En nuestro caso, describimos la morfología binaria, que no es sino un caso particular de la morfología matemática en la que las imágenes de entrada solo presentan dos posibles valores.

5.2.2.1 RELACIONES ENTRE PIXELES:

Un píxel p de coordenadas (x,y) tiene cuatro vecinos horizontales y verticales cuyas coordenadas vienen dadas por:

[(x+1,y), (x- 1,y), (x, y+1), (x,y -1)]

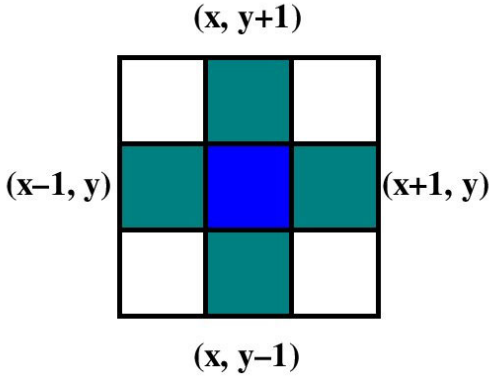


Fig 5.4 Contorno de orden 4

Este conjunto de pixeles, denominado los 4-vecinos de p, se representa por $N_4(p)$. Algunos de los vecinos de p caen fuera de la imagen digital si (x,y) está en el borde de la imagen.

Los cuatro vecinos en diagonal de p tienen las coordenadas:

(x+1, y+1), (x+1, y- 1), (x-1, y +1), (x-1, y-1)

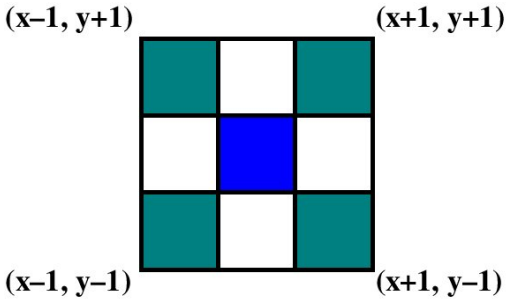


Fig 5.5 Contorno de orden 4 diagonal

y se representan por ND(p).

Estos puntos, junto a los 4-vecinos, se denominan los 8-vecinos de p, y se representan por N8(p). Al igual que antes, algunos puntos de ND(p) y N8(p) caen fuera de la imagen si (x,y) está en el borde de la misma.

La ***conectividad*** entre pixeles [PETM1996] es un concepto importante empleado para establecer los límites de los objetos y los componentes de áreas en una imagen. Para determinar si dos pixeles están conectados deben determinarse si son adyacentes en algún sentido (como ser 4-vecinos). Se consideran tres tipos de conectividad:

a) 4-conectividad: Dos pixeles p y q están 4-conectados si q pertenece a $N_4(p)$.

b) 8-conectividad: Dos pixeles p y q están 8-conectados si q pertenece a $N_8(p)$.

c) m-conectividad (conectividad mixta): Dos pixeles p y q están m-conectados si:

i) q pertenece a N4(p), o bien

ii)q pertenece a ND(p) y, además el conjunto $N4(p) \cap N4(q)$ es vacío.

Por consiguiente, un píxel p se considera ***adyacente*** a un píxel q si ambos están conectados.

5.2.2.2 ELEMENTO ESTRUCTURAL

El elemento estructural determina los detalles del efecto producido en la imagen por la transformación morfológica. Consiste en un patrón que especifica las coordenadas de un numero discreto de puntos relativos a un origen [ADWE1996]. La figura 5.5 muestra distintos elementos estructurales de varios tamaños, el origen está señalado por un circulo

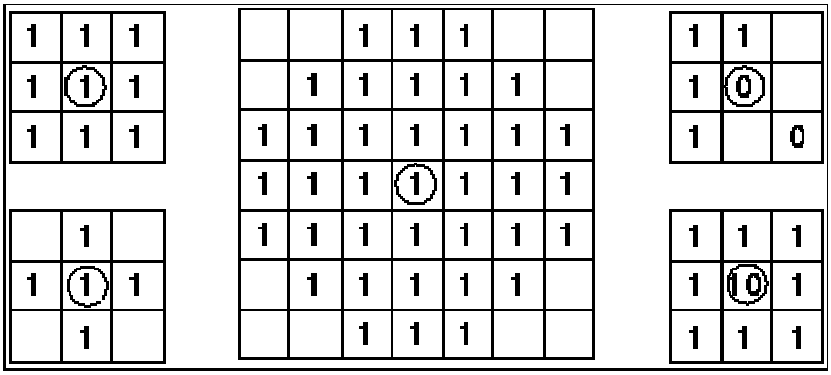


Fig 5.6 Ejemplos de elementos estructurales.

Cada punto del elemento puede tener distinto valor, generalmente, los elementos estructurales utilizados con imágenes binarias para operaciones como la erosión, solo tienen un valor, representado como 1. en otros tipos de aplicaciones, pueden tener otros valores.

En las operaciones morfológicas, el origen del elemento estructurales se traslada a cada píxel de la imagen, donde el valor de los puntos del elemento se comparan con el valor de aquellos pixeles adyacentes indicados en el elemento. El resultado de la comparación dependerá de los valores de los puntos del elemento y de los pixeles adyacentes en cada caso particular. De este modo, las coordenadas de los puntos del elemento estructurales, podrán ser tanto valores positivos como negativos.

5.2.2.3 EROSIONADO MORFOLÓGICO

Como ya hemos comentado, la erosión es una de las operaciones básicas de la morfología matemática, que puede aplicarse a imágenes en escala de grises, pero nos interesaremos en el caso de imágenes binarias, ya que el tiempo de computo será menor y obtendremos el resultado que queremos.

Básicamente, el efecto de este operador es el de “limar” los bordes de las figuras, elimina los puntos aislados y aumenta el tamaño de los “agujeros”, como se aprecia en la figura 5.7:

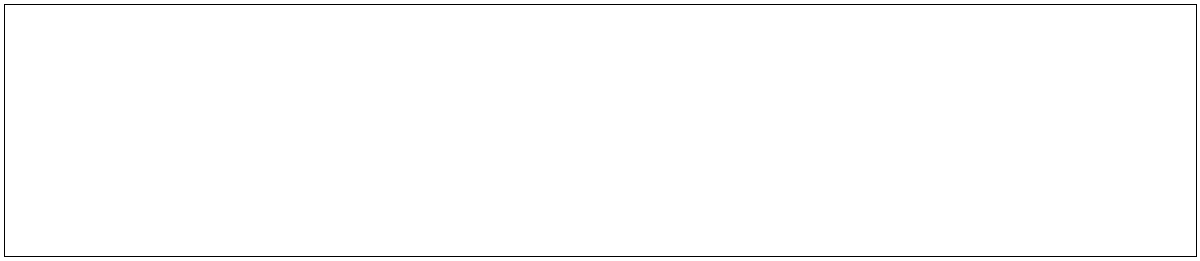


Fig 5.7 Efecto de erosión

La definición matemática de erosión para imágenes binarias [REK1994]:

1. Suponemos que X es el conjunto de coordenadas euclídeas correspondientes a una imagen binaria de entrada y K es el conjunto de coordenadas del elemento estructural.
2. Kx denota la translación de K, tal que su origen sea x.
3. Entonces, la erosión de X por K es el conjunto de todos los elementos x tal que Kx es un subconjunto de X.

En nuestro caso, el resultado que nos interesa de esta operación, puede conseguirse mediante un elemento estructural de tamaño 3x3 con el origen en el centro, uno de los tipos de elementos más común utilizado:

1	1	1
1	**1**	1
1	1	1

Conjunto de puntos: {(-1,-1),(0,-1),(1,-1), (-1,0),(1,0),(-1,1),(0,1),(1,1)}

Centro: (0,0)

Y el efecto que consigue: (En este caso, hemos representado los puntos del entorno con valor 0, y los puntos de interés con valor1).

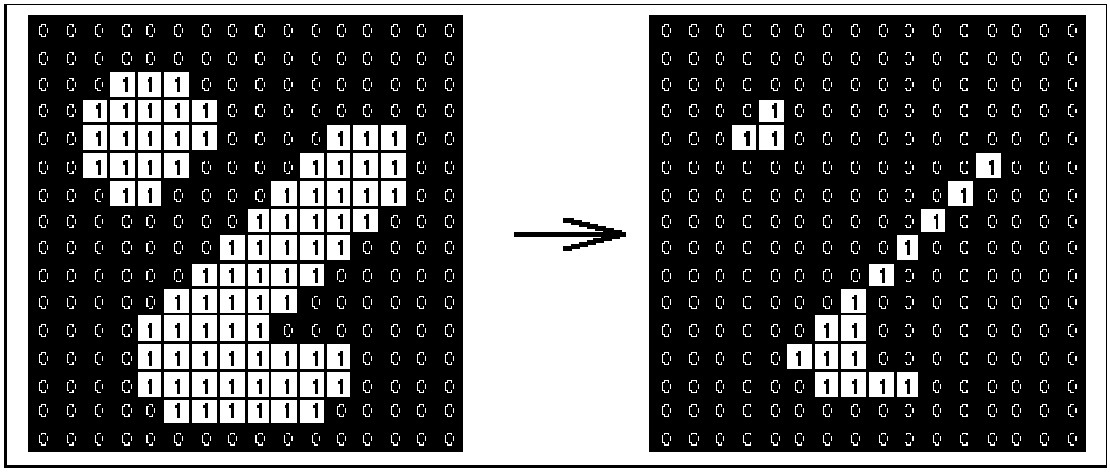


Fig. 5.8 Valores de pixeles tras erosión binaria

El algoritmo de erosión, con el elemento estructural que hemos descrito, puede resumirse en los siguientes pasos:

1. Localizar los puntos de la imagen de valor 1 (o 255, según la implementación), a excepción de los que se encuentran en el borde de la imagen.
2. Para cada punto de valor 1, comprobar si *todos* los puntos adyacentes a él tienen valor 1. (Para imágenes en escala de grises, se busca el valor mínimo de los pixeles adyacentes al central).
3. En caso afirmativo, el píxel conserva su valor 1, en caso contrario recibe valor 0 (y pasa por tanto a ser un píxel externo).(Para imágenes en escala de grises, el píxel central toma el valor mínimo de los adyacentes).

5.3 DESARROLLO DEL ALGORITMO

5.3.1 UMBRALIZACIÓN:

El primer paso que debemos llevar a cabo, es el de umbralizar la imagen, separando así los puntos que coinciden con el rango de colores de la bola, del resto de la imagen.

Para ello, debemos realizar un análisis previo de la imagen y registrar los rangos de colores de toda la imagen sin bola, para compararlo con el contenido de los puntos de la bola, y encontrar así un umbral óptimo de filtrado.

Tomamos una imagen de prueba representativa del 'peor caso', es decir, una imagen como la de la figura 5.9, muy sobrecargada de elementos y colores:

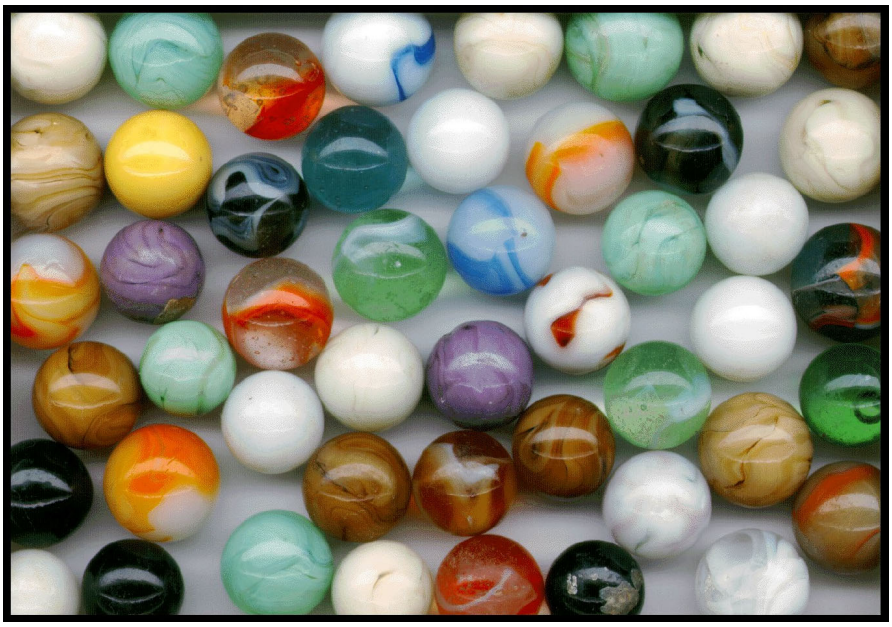
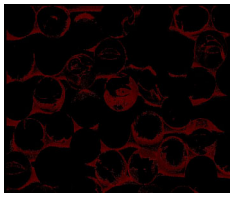
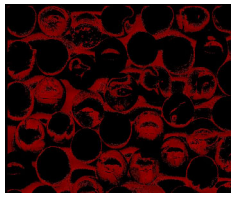
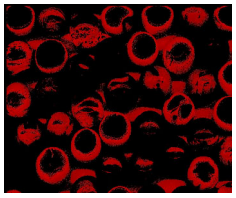
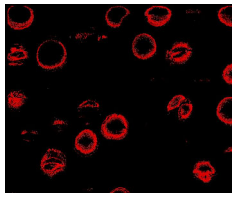
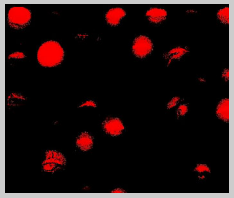
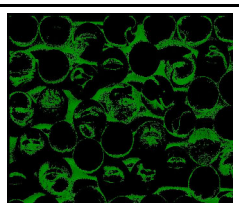
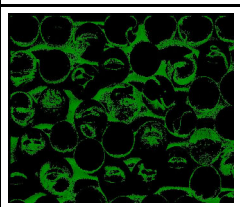
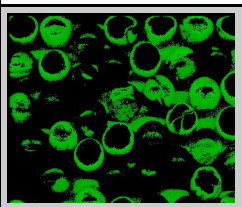
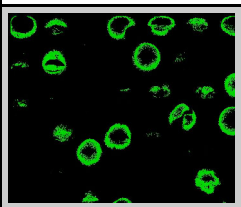
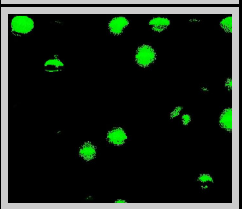


Fig. 5.9. Imagen de prueba

A continuación observamos los rangos de contenido de cada color, estos rangos, constituirán la máscara de umbralizado. Ya que en la aplicación final utilizaremos una canica de color amarillo, vamos a describir el proceso para la bola de este color.

Para encontrar los rangos de filtrado, podemos utilizar el histograma de la imagen, sin embargo, dadas las características de esta imagen, un método más 'visual', es observar tramos en cada color. Como puede verse en la figura 5.10, el color que buscamos tiene alto contenido en rojo y en amarillo, pero no en azul.

	0-100	***100-150***	***150-200***	***200-225***	***225-256***
R O J O					
V E R D E					

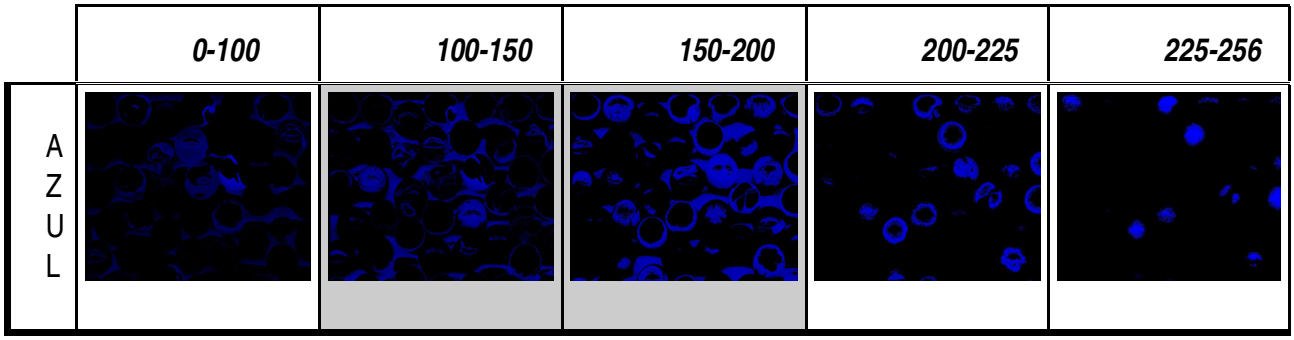


Fig 5.10 Distribución de colores de la imagen

Para este caso en concreto, aplicaremos unos umbrales de valor Rojo > 230, Verde> 180 y Azul < 170, conseguimos un resultado como el de la figura 5.11:

Fig 5.11 Resultado de la umbralización

Para obtener una imagen como la de esta figura, en lugar de una binarización, lo que se hizo fue, en cada campo de color, almacenar un cero si el píxel no pertenece a la bola, y el valor original del píxel si este era interno a la bola (es decir, entre *umbral* y 255, o entre 0 y *umbral*). Para realizar pruebas, este tipo de imagen es mas significativa que una imagen binaria, ya que al poder reconstruir el color verdadero, podemos comprobar que estamos seleccionando las zonas correctas.

En la aplicación final, no es necesario almacenar tanta información. Sin embargo, es necesario mantener una variable de tipo imagen (es decir, una que contenga los cuatro campos, rojo, verde, azul y alfa, en lugar de uno solo), ya que necesitaremos repetir la búsqueda de los

umbrales en la implementación final (los valores de RGB que guarda una imagen no coinciden en todas las implementaciones, hay que recordar que las imágenes finales se codificarán a través de la tarjeta) . La imagen binaria que trataremos asignará valor 0 a los puntos del entorno y 255 a los puntos de interés, debido a que visualmente, estos valores contrastan mas.

Una vez encontrados los umbrales, para aligerar el proceso, la función de filtrado devolverá la imagen binaria en uno de los campos, en nuestro caso en el campo Verde, dejando sin utilizar a partir de ese momento los otros campos. Se eligió el verde, por que el efecto del valor 255 de verde sobre el fondo negro de los pixeles a 0, es muy parecido a blanco sobre negro y es muy fácil distinguir los pixeles, mientras que los valores 255 de rojo y azul aparecen como gris.

De esta manera, una vez comprobado que esta solución no presenta problemas de memoria (cada imagen ocuparía 320*240*4 bytes, 300 KBytes), todas las funciones intercambiarán el mismo tipo de datos que las librerías bttv_marte, y en cualquier fase del proceso podremos ver por pantalla la imagen que estamos procesando.

Es decir, a partir de la etapa de filtrado trataremos con una imagen binarizada como la de la figura 5.12:

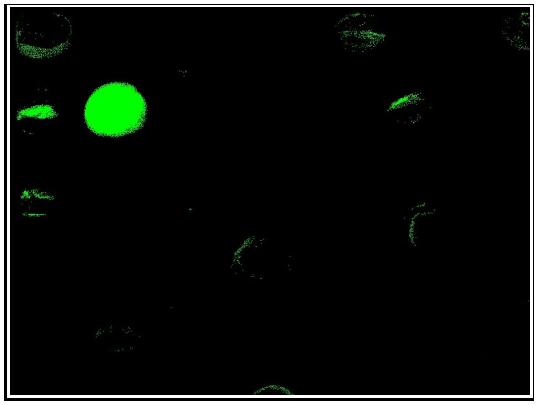


Fig. 5.12 Imagen binarizada

5.3.2 EROSIONADO:

Con el objetivo de mejorar el resultado obtenido en la etapa de segmentación, eliminaremos los puntos de la imagen cuyo rango de colores coincide con el de la bola mediante la técnica de erosionado.

Para implementar este proceso, recorremos todos los pixeles pertenecientes a la bola. Se mantendrán como puntos de interés (con valor 255) aquellos puntos que no estén rodeados por ningún punto externo, mientras que el resto se considerarán puntos del fondo (a valor 0).

En la figura 5.13 se observa el resultado de erosionar el canal verde de nuestra imagen de ejemplo, ampliando la zona de interés:

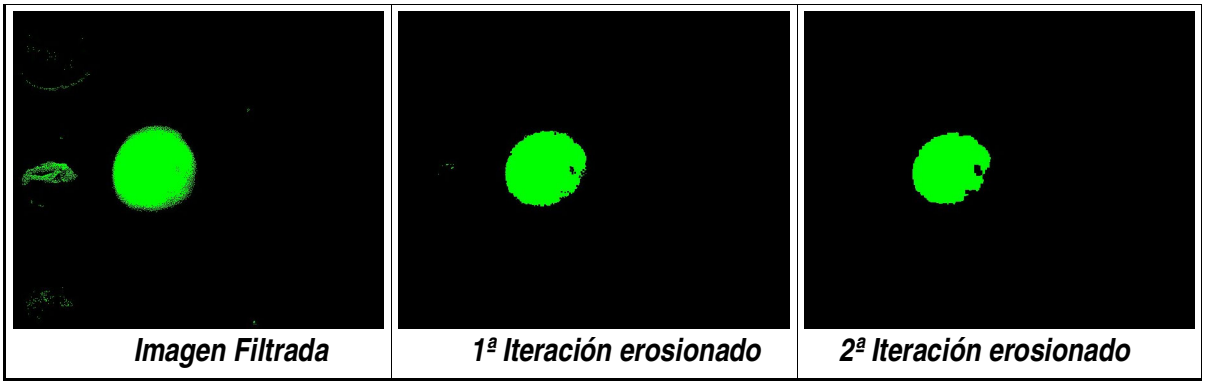


Fig 5.13 imagen erosionada

En este caso, dos iteraciones de erosionado son suficientes para eliminar todos los puntos no deseados.

5.3.3 BÚSQUEDA DEL CENTRO:

En las etapas anteriores, hemos conseguido segmentar la imagen, con este resultado, devolveremos, como descripción del objeto buscado, su centro de masas.

A partir de los puntos obtenidos tras la etapa de erosión, hallaremos el centro de la bola. No podemos simplemente hallar la media de las x e y mínimas y máximas, porque la existencia de puntos del fondo no eliminados invalidaría este resultado. Sin embargo, podemos hallar una buena aproximación del centro hallando la media en ambos ejes de todos los puntos ya que el porcentaje de puntos exteriores es muy bajo en comparación con el de puntos válidos.

En la figura podemos ver los resultados del centro obtenido a partir de una imagen filtrada y erosionada, el centro calculado en cada caso se representa por un punto:

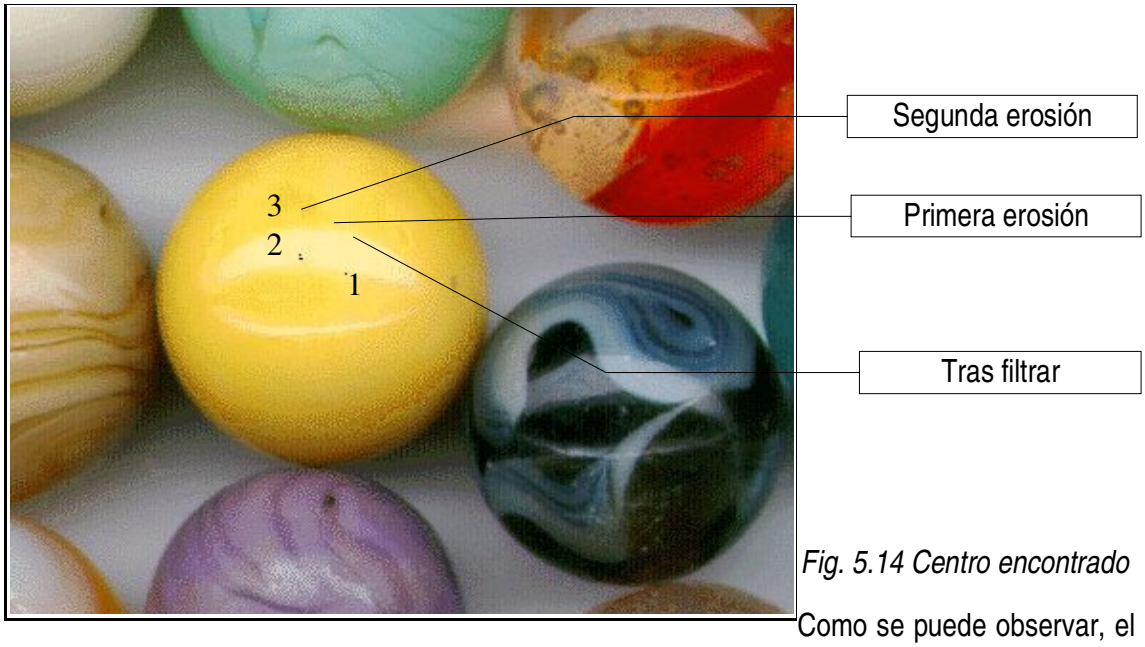


Fig. 5.14 Centro encontrado

Como se puede observar, el centro al que converge el proceso no coincide con el que buscamos. Si superponemos sobre la imagen original los puntos que conservamos tras el filtrado (fig. 5.15), vemos que, si bien la mayoría de puntos externos fueron eliminados, también se elimino gran parte de los puntos de la bola.

Como es lógico, el centro que calculamos converge al centro de los puntos que se conservan tras el filtrado. Si observamos la imagen original, comprobamos que los puntos de la bola eliminados en el filtrado pertenecen a zonas con sombra, por eso habrá que tener muy en cuenta la iluminación en el diseño final de nuestro montaje.

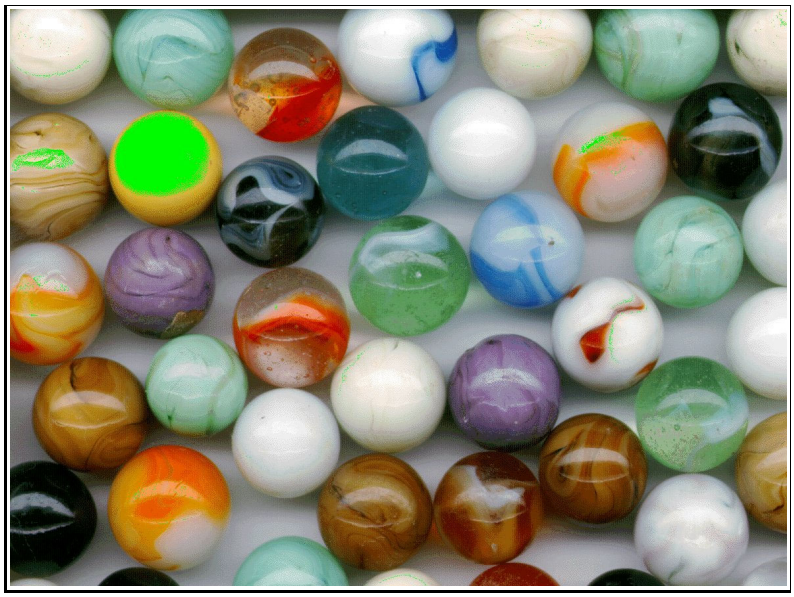


Fig. 5.15 Superposición umbralizado (agresivo) sobre imagen original

Lo que hemos hecho, es eliminar el mayor numero de puntos externos, motivo por el cual es necesaria una sola iteración de erosionado.

Para encontrar el centro de esta bola en esta imagen sobrecargada, deberíamos realizar un filtrado que conservase el mayor numero de puntos de la bola, para eliminar los externos a continuación mediante varias iteraciones de erosionado. En la figura 5.16 podemos observar el resultado de aplicar una etapa de filtrado menos “agresiva”:

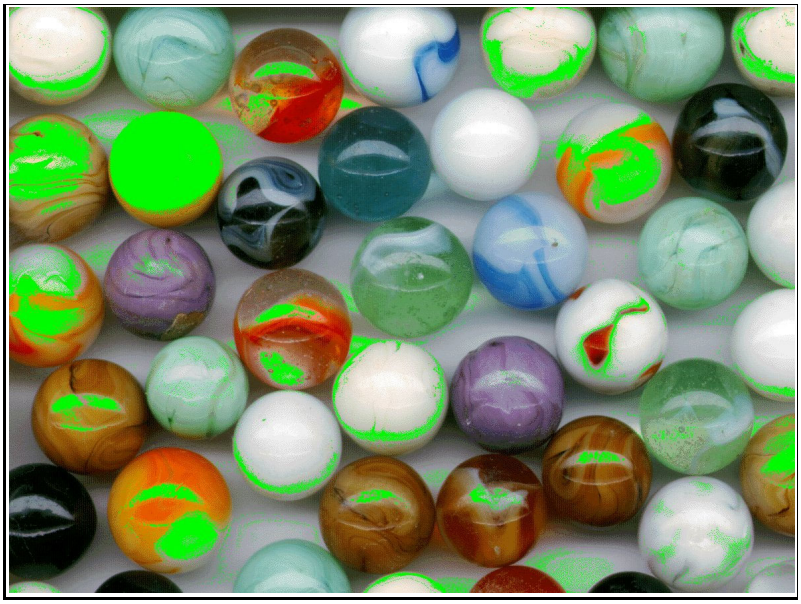
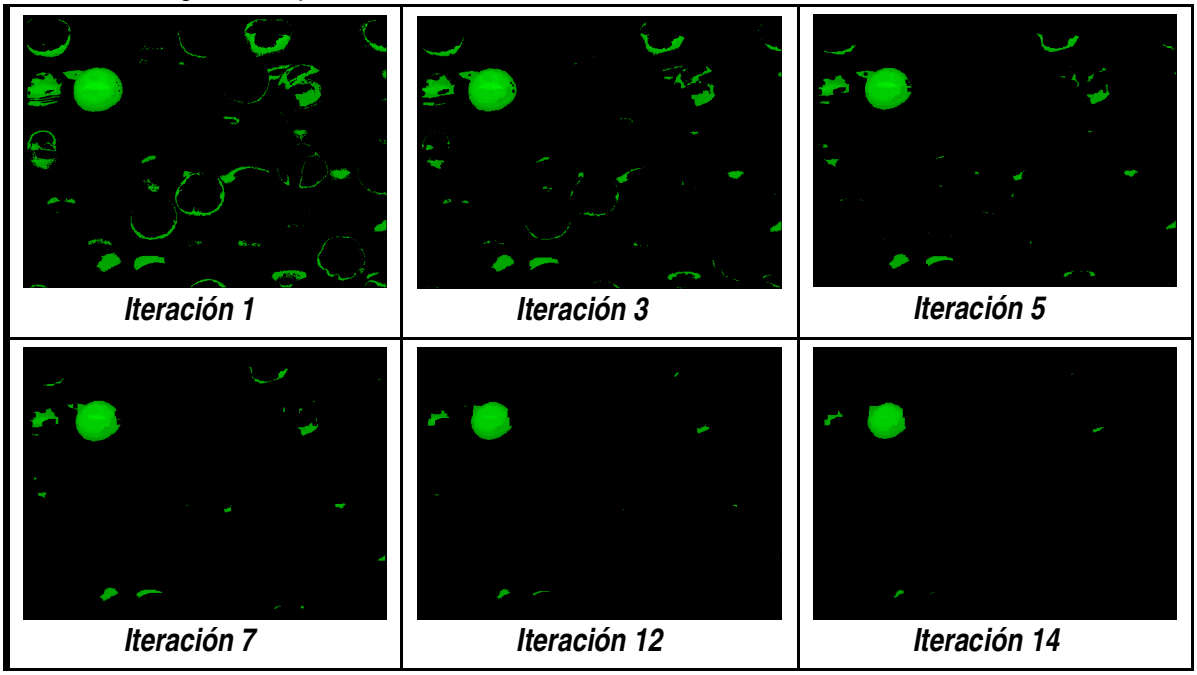


Fig 5.16 Superposición umbralizado (suave) sobre imagen original

En este caso son necesarias 20 iteraciones de erosionado para converger al centro de la bola. en la figura 5.17 podemos ver sucesivas iteraciones de erosionado:



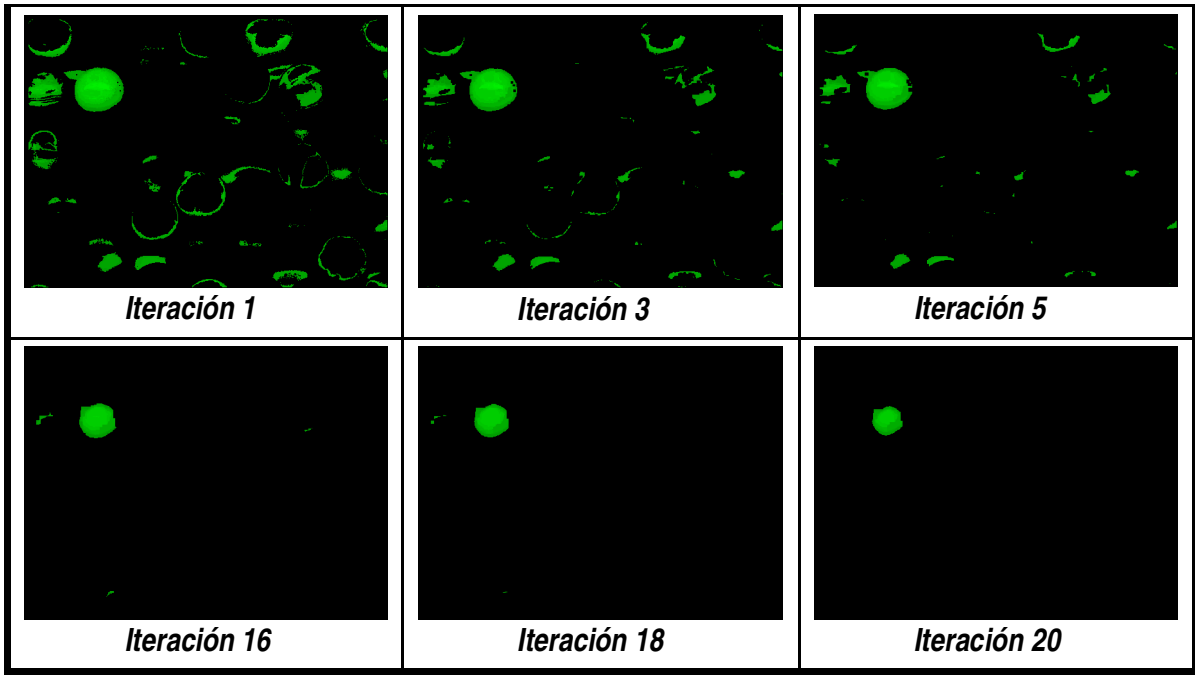


Fig. 5.17 Iteraciones de erosionado

Podemos ver como converge hacia el centro a medida que erosionamos en la figura 5.17:

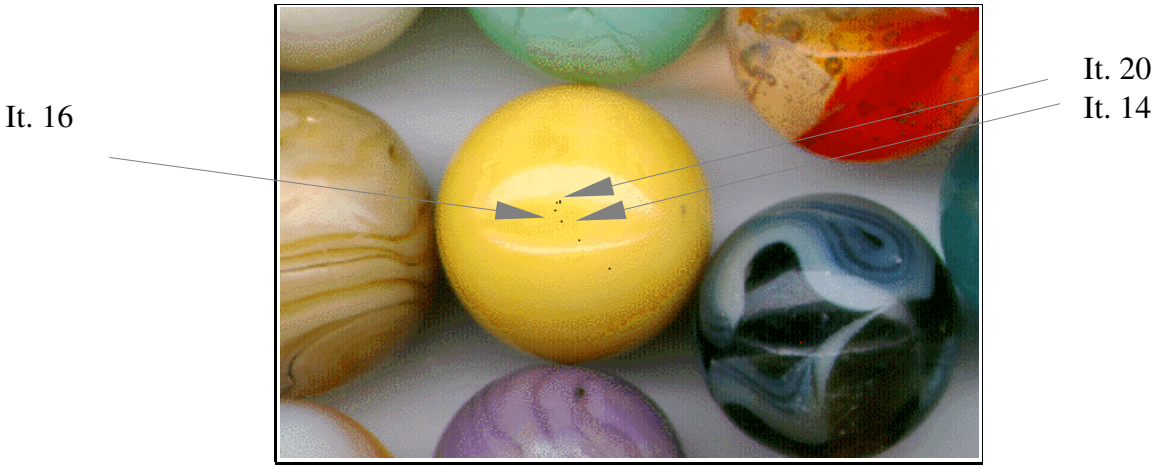


Fig. 5.17 Convergencia del centro

Sin embargo aún se aprecia que el centro al que converge, no tiene en cuenta la parte sombreada de la bola. Este fue el motivo principal por el que se añadieron los focos al sistema, por lo que estas sombras no aparecerán y el cálculo del centro será más exacto. Además, los focos independizan al conjunto de la luz exterior. Supongamos el caso de que, por cualquier motivo, trasladásemos la estructura a otra ubicación, en la que la luz llega desde un lateral, el procesado de imagen no sería uniforme en toda la plataforma, si no que dependería del punto en que la bola se encuentra.

La primera solución era mas rápida, ya que, el erosionado es la tarea de mayor carga computacional. Con el filtrado agresivo, en un ordenador *Pentium 4 a 2.4 Ghz,* obtenemos un centro cada 112 ms (promedio de 100 repeticiones y teniendo en cuenta el retraso de la librería Gandalf).

Sin embargo, el proceso con un filtrado mas suave, permite resultados más precisos. En 825.5 ms obtendríamos el centro con 20 iteraciones de erosionado. El incremento de tiempo en las sucesivas iteraciones se representa en la gráfica 5.19, promediando de 100 repeticiones, para imágenes de 640x480 pixeles:

Iteración	*ms.*
1	*111.89*
2	*158.1*
4	*217.7*
8	*371.6*
12	*523.2*
16	*671.25*
20	*825.5*
30	*1.271(sg)*
40	*1.664(sg)*

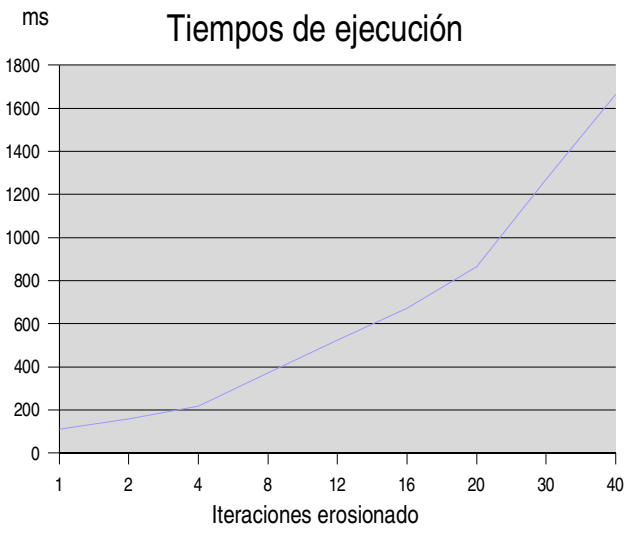


Fig 5.19 Tiempos de ejecución del erosionado

Según la aplicación, tendremos que llegar a un compromiso ente precisión y tiempo. En nuestro caso, la imagen que vamos a tratar no estará tan sobrecargada de objetos, por lo que podremos encontrar el centro de la bola con pocas iteraciones de erosionado. Este centro será preciso siempre y cuando el color de la bola sea lo más uniforme posible, es decir, sin brillos ni sombras.

5.3.4 MEJORAS:

Una mejora posible, seria corregir la primera aproximación del cálculo del centro, eliminando aquellos puntos externos a la bola que no hayan sido previamente eliminados. A partir de este centro calculado, serán puntos exteriores a la bola todos aquellos cuya distancia al centro de la bola sea mayor que el radio de esta.

En la práctica, esta distancia se multiplicara por un factor K>1, para compensar el error introducido en el calculo anterior del centro, ya que un valor de radio muy pequeño podría suponer que eliminásemos puntos validos. A continuación se recalcula la posición del centro de la bola.

En nuestro caso, la imagen final no contendrá puntos externos una vez terminado el proceso de segmentación, como veremos más adelante. Por este motivo, la función que implementa esta búsqueda, permite decidir el nivel de precisión deseado, es decir, si se desea como valor de centro la primera aproximación o el valor corregido, para hacer el proceso mas ligero y eficiente.

5.3.4.1 RECORTADO:

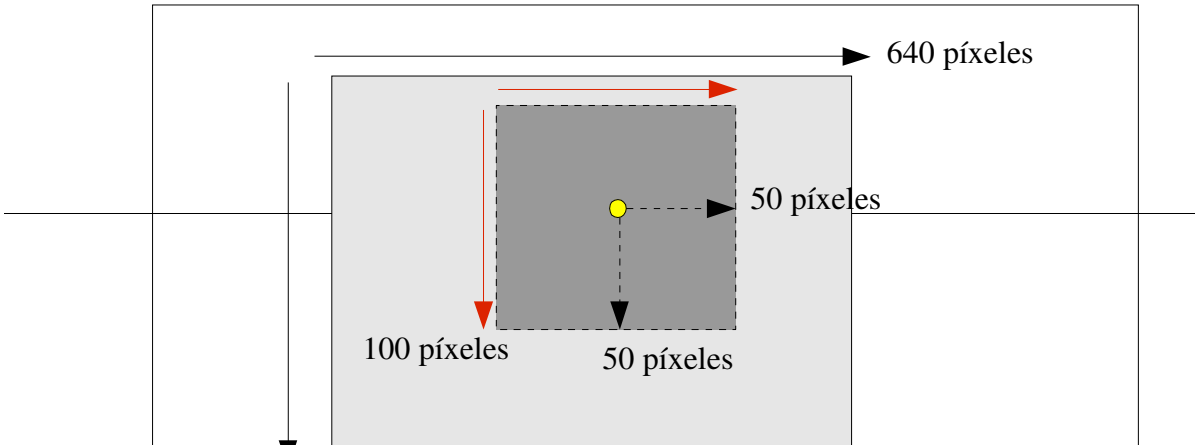
Solo en la primera iteración es necesario procesar toda la imagen. En el resto, será suficiente con procesar únicamente un entorno de la ultima posición conocida de la bola. Este entorno, deberá ser algo mayor que la máxima distancia que la bola puede recorrer en un periodo de tiempo, para compensar errores de cálculo de la posición del centro, y para prever retardos en la ejecución de las tareas.

A partir de la velocidad máxima que la bola alcance en el plano, podemos conocer la distancia máxima en pixeles que es capaz de recorrer en 20 ms.

La máxima velocidad que la bola puede alcanzar, es de unos 30 cm/sg. La cámara se colocará para captar el tablero completo con la menor superficie externa a él, luego los 480 pixeles (la menor de las dos dimensiones) se ajustarán para cubrir los 30 cm de tablero. Es decir, si recorre 480 pixeles en 1 segundo, recorre 9.6 pixeles por período.

Con este razonamiento, de cada imagen, procesaremos ventanas de 100x100 pixeles, es decir, la bola debería llevar cinco veces la velocidad máxima para abandonar la ventana de visión. De este modo, en caso de que exista algún retraso en las rutinas de control, será poco probable no encontrar la bola.

La figura 5.20 muestra un esquema de la ventana abierta a partir de la última posición conocida de la bola.



480 pixeles

Fig.21 ventana de 100x100 pixeles en la imagen

De este modo, las funciones de filtrado y erosión se aligeran considerablemente al procesar una imagen mucho menor, como se describe en el apartado 5.4.3.

5.3.4.2 DIEZMADO:

Otra solución para aligerar el proceso sería procesar solo una parte de los pixeles, en vez de la imagen completa. Para ello, podemos *muestrear* la imagen, es decir, en vez de una imagen de 640x480, podemos obviar la mitad de los pixeles de cada fila y procesar una de 320x240, es decir, diezmar por un factor 2.

La diferencia con la solución anterior, es que en este caso, se analizan pixeles de toda la imagen, por lo que la bola queda siempre visible. Sin embargo, la calidad de esta imagen es menor, ya que, diezmar por un factor 2 implica descartar 3 de cada 4 pixeles, por un factor 3, se descartan 8 de cada 9 y así sucesivamente.

En la figura 5.21 se observa la pérdida de calidad de la imagen a medida que aumenta el factor de diezmado (hemos ampliado la zona de interés para percibir mejor el efecto):

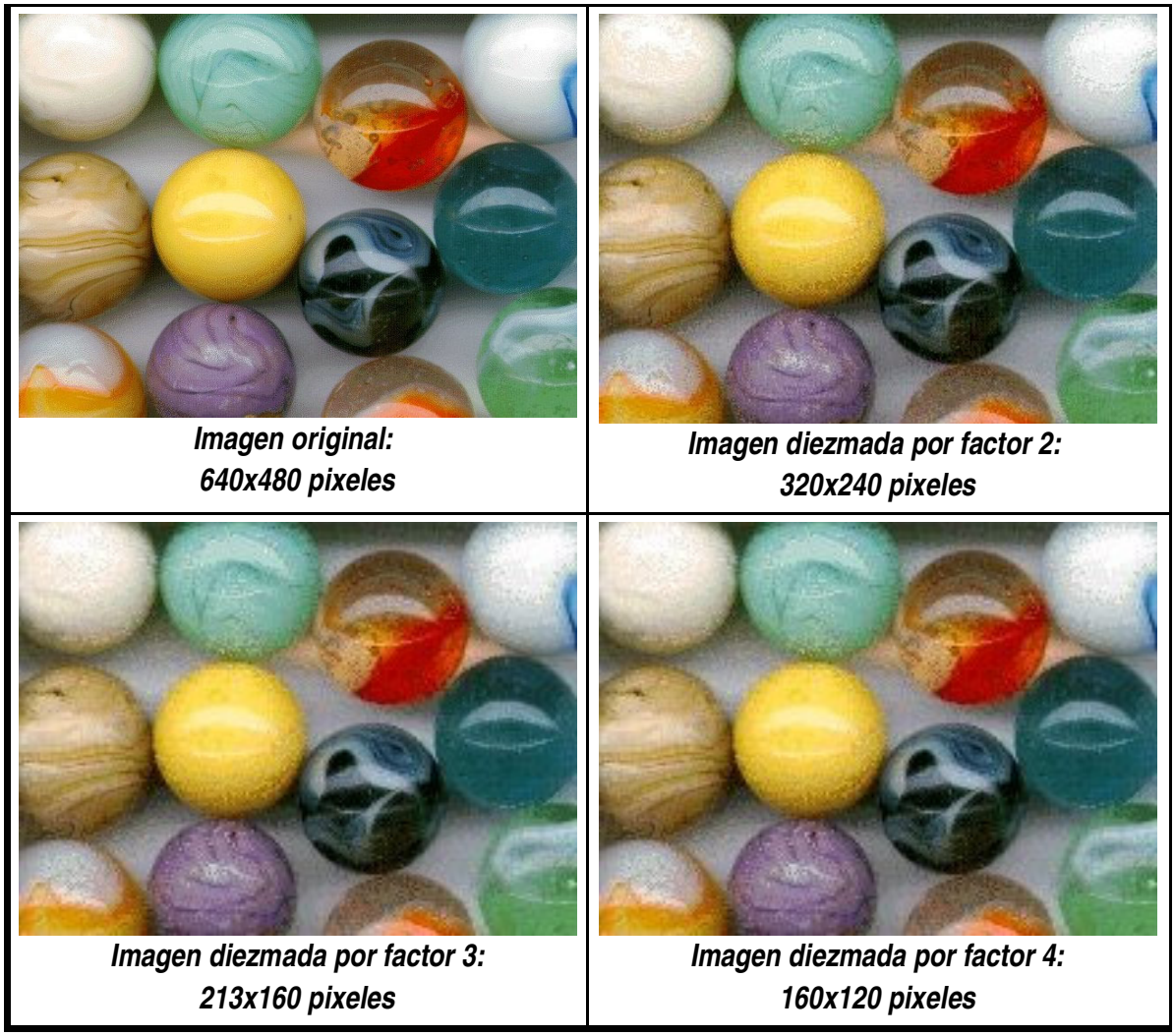


Fig 5.20 Efecto del diezmado

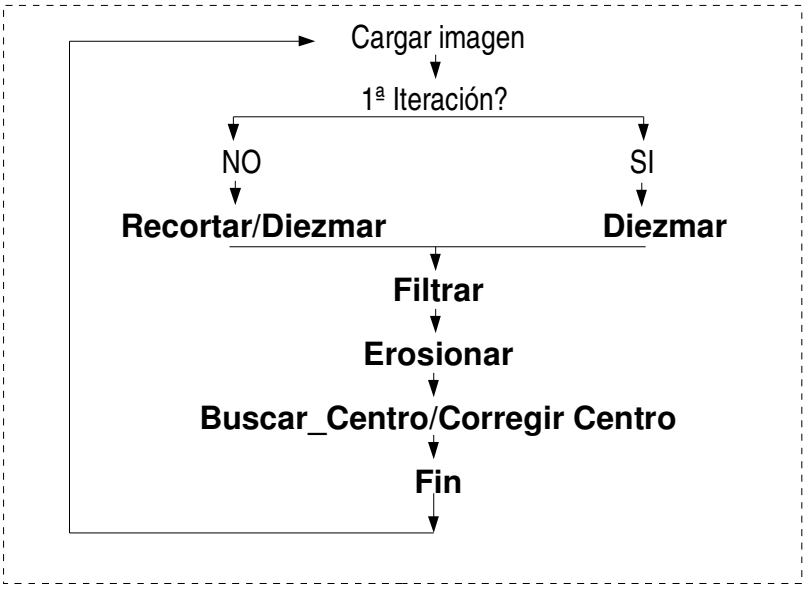
Estas imágenes de menor calidad pueden utilizarse en aquellas aplicaciones en las que la calidad del resultado obtenido no sea tan importante como obtener una estimación rápida de la posición. En nuestro caso, para realizar un procesado de la imagen completa en la primera iteración del proceso, donde es necesario recorrer toda la imagen, ya que la posición nos es por completo desconocida.

En la solución final adoptada para MaRTE, debido al problema con el barrido entrelazado, las imágenes son de 320x240, por lo que las imágenes no se diezmaran. Sin embargo, la librería desarrollada para Linux permite escoger el factor de diezmado deseado.

Las operaciones de recortado y diezmado de la imagen, se incluirán como parte del preprocesado de la misma, al introducir mejoras que permitirán incrementar, en cuanto a tiempo de ejecución, en las etapas siguientes del proceso.

5.4 LIBRERÍA DE IMAGEN:

El esquema del proceso completo disponible para Linux puede resumirse en el siguiente esquema:



5.4.1 TIPOS DE DATOS:

- Estructura con los colores del punto

```
struct punto{
  int     r;
  int     g;
  int     b;
};
```

- Estructura con las coordenadas del punto

```
struct xy{
  int     x;
  int     y;
};
```

- Tipo en el que se guarda el histograma

```
typedef struct{
  int     r;
  int     g;
  int     b;
} hist [256];
```

- Tipo en el que se guarda una imagen RGB de tamaño máximo 1200x1200 pixeles:

```
typedef struct{
  int     r;
  int     g;
  int     b;
} matriz[1200][1200];
```

5.4.2 DESCRIPCIÓN DE FUNCIONES IMPLEMENTADAS:

A continuación se describen las funciones implementadas:

- **Hallar un histograma:**

```
int crea_histograma (matriz      foto,
                     struct xy   *tamano,
                     hist        histograma)
```

Esta función permite obtener el histograma de la imagen de entrada, del tamaño indicado por *Tamano*. Está pensada para imágenes RGB, devolviendo por separado los tres histogramas, pero podría utilizarse para imágenes en escala de grises, pasando la imagen en uno de los campos.

Devuelve un entero de valor cero si se ejecuta sin errores, y menor que cero en caso contrario con los siguientes códigos de error:

- (-81) : Tamaño no válido; tamaño negativo en x o y.

- **Cargar una imagen:**

```
int carga_foto_ppm (char      *nombre,
                    matriz    foto,
                    struct xy *tamano)
```

Esta función carga una foto con formato PPM 6 en una variable de tipo *MATRIZ,* con la que operan el resto de funciones. El tamaño que recibe la función debe ser menor o igual que el de la

imagen original, en caso de ser menor, se carga solo el tamaño indicado desde la esquina superior izquierda.

La función utiliza a su vez la librería gráfica Gandalf, para acceder a la información de la imagen. Devuelve un entero de valor cero si se ejecuta sin errores, y menor que cero en caso contrario con los siguientes códigos de error:

- (-11) : Tamaño no válido; tamaño negativo en x o y. Si el tamaño es mayor que el de la imagen original, Gandalf eleva una excepción.

■ **Obtener un recorte de imagen:**

```
int recorta( matriz      foto,
             matriz      foto_recortada,
             struct xy   tamano,
             struct xy   *tamano_nuevo,
             struct xy   centro,
             int         d_max,
             struct xy   * origen_nuevo);
```

Esta función recibe una imagen en la variable *foto,* de tamaño *tamano. R*ecibe también la distancia máxima en pixeles que la bola ha podido recorrer desde el instante anterior, en la variable *d_max*, a partir del último centro conocido, que se recibe en la variable *centro*.

Devuelve un recorte de tamaño menor o igual que (2**d_max*)*(2**d_max*). En caso de que el centro se encuentre a una distancia menor que *d_max* de los extremos de la imagen en ambos ejes, el tamaño devuelto en la variable *tamano_nuevo* será 2**d_max* (Fig 5.22 a).

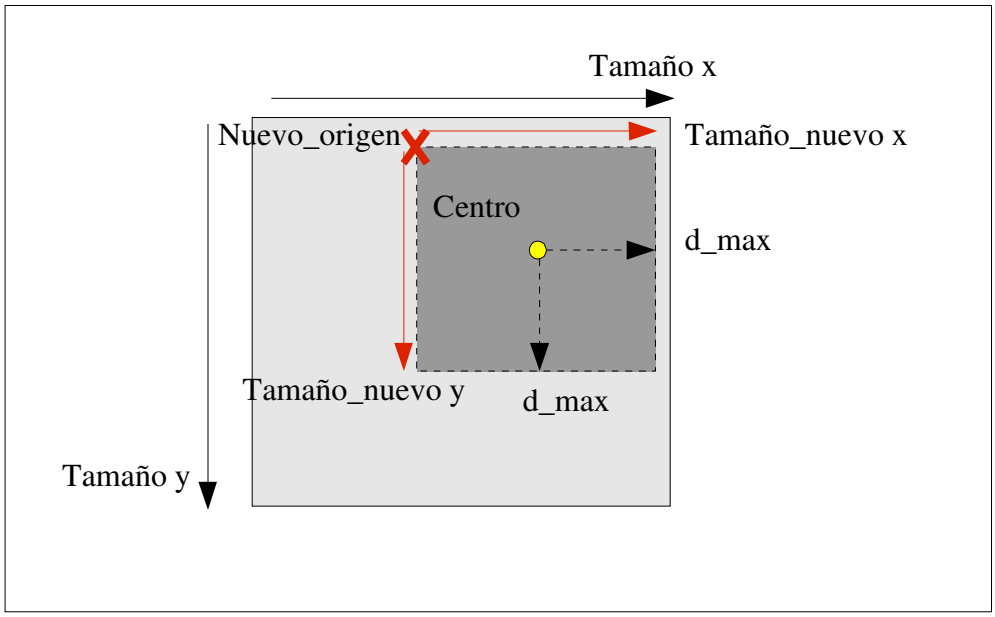


Fig 5.22 a) centro a distancia menor que d_max de los extremos

Si el centro se encuentra a una distancia menor que *d_max* del extremo de la imagen, se devuelve la distancia del nuevo origen hasta el tamaño de la imagen:

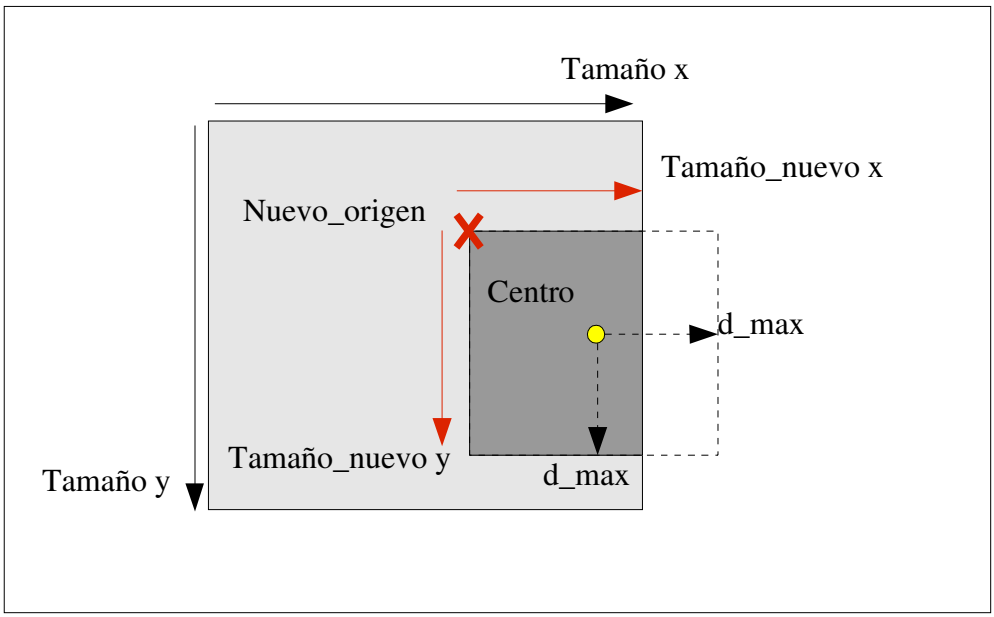
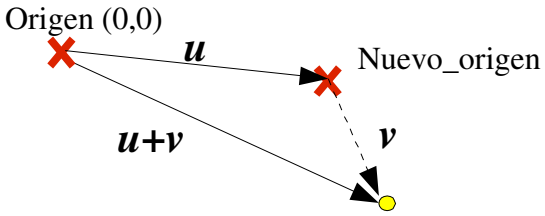


Fig 5.22 b) centro a distancia mayor que D_MAX de los extremos

La función devuelve las coordenadas del nuevo origen de coordenadas, de este modo, podemos ubicar el recorte en la imagen original. Las coordenadas de un punto del recorte pueden traducirse inmediatamente a coordenadas en la imagen original con solo aplicar la siguiente relación vectorial:

Sea u la posición del nuevo origen respecto del original, v la posición de un punto respecto del nuevo origen, podemos escribir las coordenadas del punto respecto del origen como:

P = u + v



La función devuelve un entero de valor cero si se ejecuta sin errores, y menor que cero en caso contrario con los siguientes códigos de error:

- (-21) : Tamaño no válido; tamaño negativo en x o y.
- (-22) : Centro no válido; si se encuentra fuera del tamaño de la imagen.

- (-23): D_max no válida; si es menor de 1 o mayor que la imagen.

■ **Filtrar una imagen y obtener imagen binaria:**

```
int filtrado (matriz      foto,
              matriz      foto_filtrado,
              struct xy   tamano,
              struct xy   *tamano_nuevo,
              struct punto umbral_rgb,
              Int         factor_diezmado);
```

A partir de una imagen de tamaño definido por Tamaño, devuelve la imagen filtrada según los valores de la variable umbral_RGB. Permite seleccionar el factor de diezmado a través de la variable factor_Diezmado (1 si se desean todos los puntos), y en este caso devuelve en la variable Tamano_nuevo el tamaño de la imagen tras el diezmado.

Devuelve un entero de valor cero si se ejecuta sin errores, y menor que cero en caso contrario con los siguientes códigos de error:

- (-31) : Tamaño no válido
- (-32): Rango de filtrado no válido; si alguno de los umbrales RGB está fuera del rango (0,255)
- (-33): factor de diezmado no válido; negativo o mayor que el tamaño de la imagen
- (-34): Ningún punto cumple los umbrales de filtrado, ya que, en ese caso, la función de erosionado recibirá una imagen vacía.

■ **Erosionar una imagen:**

```
int erosionado (matriz    foto,
                matriz    foto_erosionada,
                struct xy tamano,
                int       iteraciones,
                int       color_erosion);
```

Devuelve en *foto_erosionada* la imagen que recibe en *foto* tras un proceso de tantas erosiones como indique *iteraciones*. El canal que se desea erosionar se escoge a través de *color_erosion (R:1,G:2,B:3).*

Devuelve un entero de valor cero si se ejecuta sin errores, y menor que cero en caso contrario con los siguientes códigos de error:

- (-41) : Tamaño no válido
- (-42): Número de Iteraciones no válido; si es menor que cero
- (-43): Color_erosion no valido; distinto de 1,2 ó 3.

Así mismo escribe un mensaje de aviso indicando la iteración en la que no quedan puntos que erosionar.

■ **Búsqueda del centro:**

```
int busca_centro (matriz      foto,
                  struct xy   tamano,
                  int         umbral,
                  int         radio,
                  int         color,
                  struct xy   *centro)
```

Devuelve el centro de todos los puntos del color indicado en color (R:1,G:2,B:3), cuyo valor sea mayor que umbral. si se desea corregir la primera aproximación de este valor eliminando todos los puntos que se encuentren a una distancia mayor que el radio de la bola, la variable *radio* debe ser distinta de 0 y del valor de radio de la bola.

Devuelve un entero de valor cero si se ejecuta sin errores, y menor que cero en caso contrario con los siguientes códigos de error:

- (-51) : Tamaño no válido; tamaño negativo en x o y.
- (-52) : Umbral no válido; si se encuentra fuera del rango (0,255)
- (-53): Radio no válido; si supera el tamaño de la imagen.

Así mismo escribe un mensaje de aviso indicando la iteración en la que no se encuentre ningún punto, o no se encuentre ningún punto externo al radio para corregir la posición del centro.

■ **Guardar una imagen:**

```
int pinta_foto(char       * nombre,
               matriz     vector,
               struct xy  tamano);
```

Utilizando la librería Gandalf, esta función guarda en una imagen de formato PPM6 el contenido de *vector*, con el nombre que se le indique.

Devuelve un entero de valor cero si se ejecuta sin errores, y (-61) en caso de obtener un tamaño no válido.

■ **Pintar un punto en una imagen:**

```
int pinta_punto ( char      * nombre,
                  matriz    foto,
                  struct xy tamano,
                  struct xy punto);
```

Esta función permite marcar en negro las coordenadas de *punto* en la imagen guardada en *foto* y a continuación guarda una copia de esta modificación con el nombre indicado. Muy útil para señalar el centro obtenido en el proceso en la imagen original.

Misma comprobación de errores que *Pinta_foto* con código de error (-71).

■ **Realizar el procesado completo de una imagen:**

```
void procesa_imagen ( int             veces ,
                      struct xy       * centro_anterior ,
                      struct xy       * centro_bola,
                      struct timespec * instante,
                      int             * salida_error   );
```

Esta función realiza el procesado completo de una imagen que carga internamente. Se le indica si se desea el procesado completo de toda la imagen con la variable veces de valor 1, o el procesado del entrono del centro_anterior con veces > 1. Devuelve así mismo el instante en que se tomó la imagen.

Devuelve un entero de valor cero si se ejecuta sin errores, y menor que cero en caso contrario con los códigos de error de las funciones carga_imagen, filtra, erosiona, busca_centro, pinta_foto y pinta_punto.

5.4.3 MEDIDAS DE TIEMPOS DE EJECUCIÓN LINUX:

Hay que tener en cuenta que estos tiempos medidos serán menores en el montaje final, ya que ahora estamos cargando la imagen a través de la librería Gandalf y después recibiremos directamente el buffer. Sin embargo, necesitamos aligerar el tiempo que tarda el proceso completo, ya que recibiremos una imagen nueva cada 20 ms.

Si atendemos al consumo de tiempo de cada tarea:

Procesado Completo (ms)	***Mínimo***	***Media***	***Máximo***
Carga_imagen	*47.26*	*47.31*	*47.46*
Filtrado	*9.85*	*10.01*	*10.09*
Erosionado	*43.88*	*47.88*	*52.15*
Búsqueda_Centro	*5.88*	*6.69*	*7.23*
Proceso completo:	***106.87***	***111.89***	***116.93***

Tabla 5.1: Tiempos de procesado imagen de prueba

Como hemos dicho, el tiempo de carga de la imagen a través de la librería Gandalf nos es indiferente ya que en nuestra aplicación final no se utilizará esta función. Sin embargo, podemos observar que, el tiempo de procesado del resto de funciones es aún muy alto.

Vemos en la tabla 5.2 a y b, los resultados analizando una ventana de la imagen según la última posición, y también diezmando el número de pixeles de la imagen:

Procesado de Imagen diezmada (ms)	***Mínimo***	***Media***	***Máximo***
Carga_imagen	*47.26*	*47.31*	*47.46*
Filtrado	*3.28*	*3.32*	*3.41*
Erosionado	*8.89*	*9.76*	*13.12*
Búsqueda_Centro	*1.44*	*1.67*	*1.92*
Proceso completo:	***60.87***	***62.06***	***65.66***

Tabla 5.2 a. Tiempos de procesado de imagen de prueba diezmada

Procesado de Ventana de 100x100 (ms)	*Mínimo*	*Media*	*Máximo*
Carga_imagen	*47.26*	*47.31*	*47.46*
Recortado_Ventana	*0.22*	*0.25*	*0.35*
Filtrado	*0.27*	*0.32*	*0.52*
Erosionado	*1.01*	*1.09*	*1.84*
Búsqueda_Centro	*0.52*	*0.58*	*0.69*
Proceso completo:	***49.28***	***49.95***	***50.68***

Tabla 5.2 b. Tiempos de procesado de imagen de prueba enventanada

Es decir, si no tenemos en cuenta el tiempo que se tarda en obtener la imagen, tenemos los siguientes resultados:

Procesado (ms):	*Mínimo*	*Media*	*Máximo*
Imagen completa de 640x480 pixeles	*59.61*	*64.58*	*69.67*
Imagen de 320x240	*13.61*	*14.75*	*18.2*
Ventana de 100x100 pixeles	*2.02*	*2.64*	*3.22*

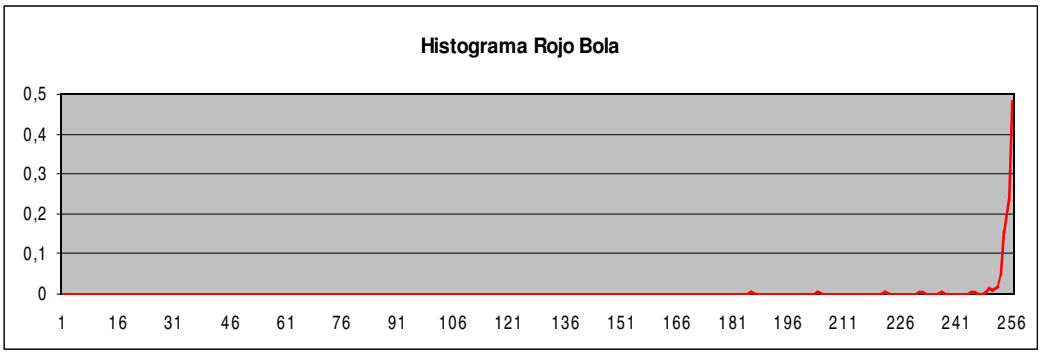
Tabla 5.2 c. Tiempos de procesado parciales

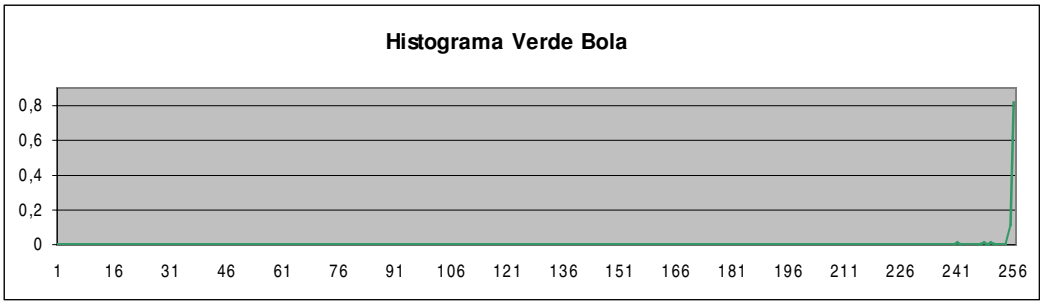
Como ya hemos comentado, en la aplicación final nuestras imágenes tendrán un tamaño de 320x240 pixeles y contendrán menos información (sólo habrá una bola), por lo que, como veremos en el apartado 5.5.2, los tiempos serán menores.

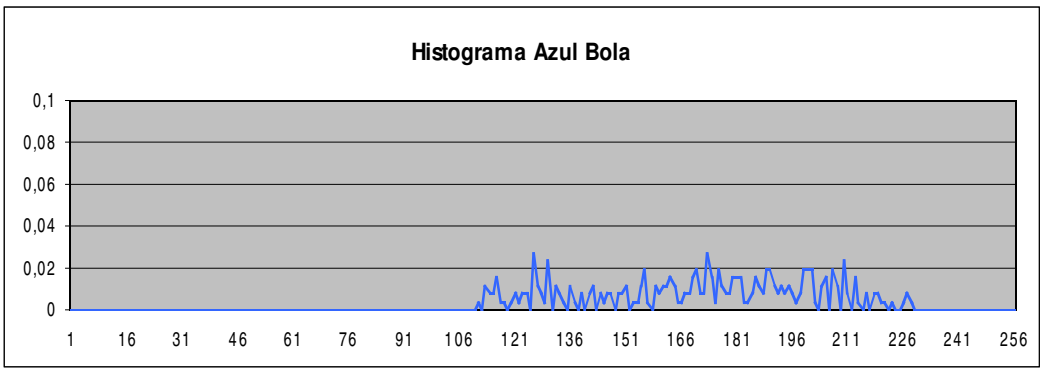
Optaremos por procesar el entorno de la última posición conocida, ya que la carga computacional aumenta de forma cuadrática con el número de puntos procesados. Sólo se procesará la imagen completa en la primera iteración o en caso de perder la bola en la ventana de imagen, y en ese caso, según el tamaño de la imagen a procesar, decidiremos si la imagen se procesa completa o diezmada.

5.5 ANÁLISIS DE IMÁGENES REALES DEL SISTEMA:

Para hallar los rangos de umbralización, partimos del histograma de la imagen completa y de la bola, comparando ambos, encontramos los umbrales óptimos para cada color:







Atendiendo al histograma de la bola, observamos que el color amarillo de la bola, se obtiene mezclando valores muy altos de verde y rojo. Las distintas tonalidades de amarillo, se consiguen variando el nivel de azul, entre 110 y 230.

En el histograma de la imagen completa, observamos dos zonas claramente diferenciadas, el entorno del valor 100-140, y la zona de valor mayor que 250. La primera zona (100-140), que no aparece en el histograma de la bola, corresponde al fondo gris de la plataforma. En la segunda zona, se observan valores muy altos, en la saturación del color, que corresponden al brillo blanco de los marcos de la plataforma. El tramo de 0 a 80 en los tres colores, corresponde al fondo negro de la imagen.

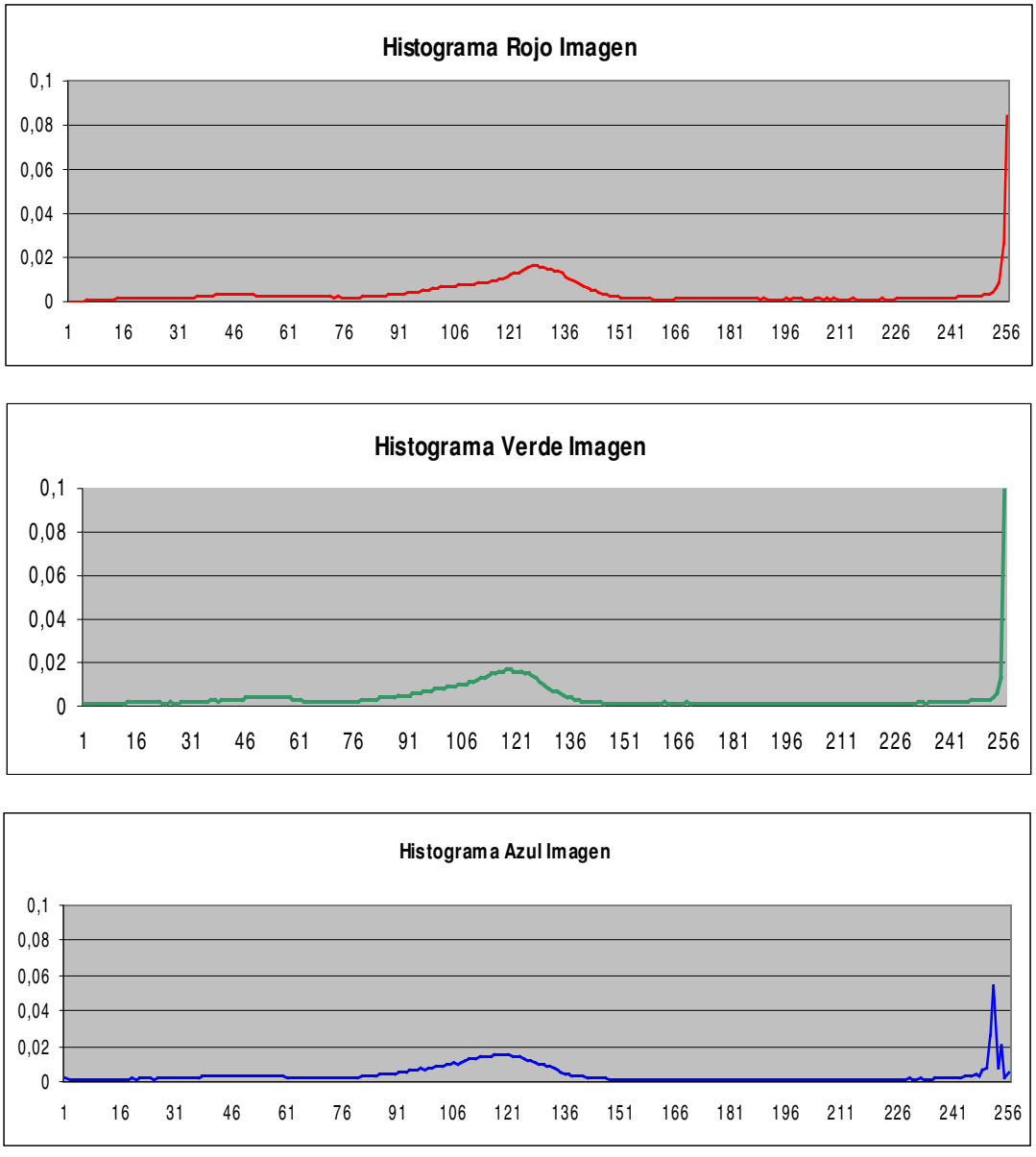


Fig 5.22 a) Histogramas imagen

Para distinguir la bola del resto de la imagen escogeremos los umbrales: R > 240, G > 240 y 110<B<230. En la figura XX observamos los distintos resultados parciales y finales del procesado de imagen:

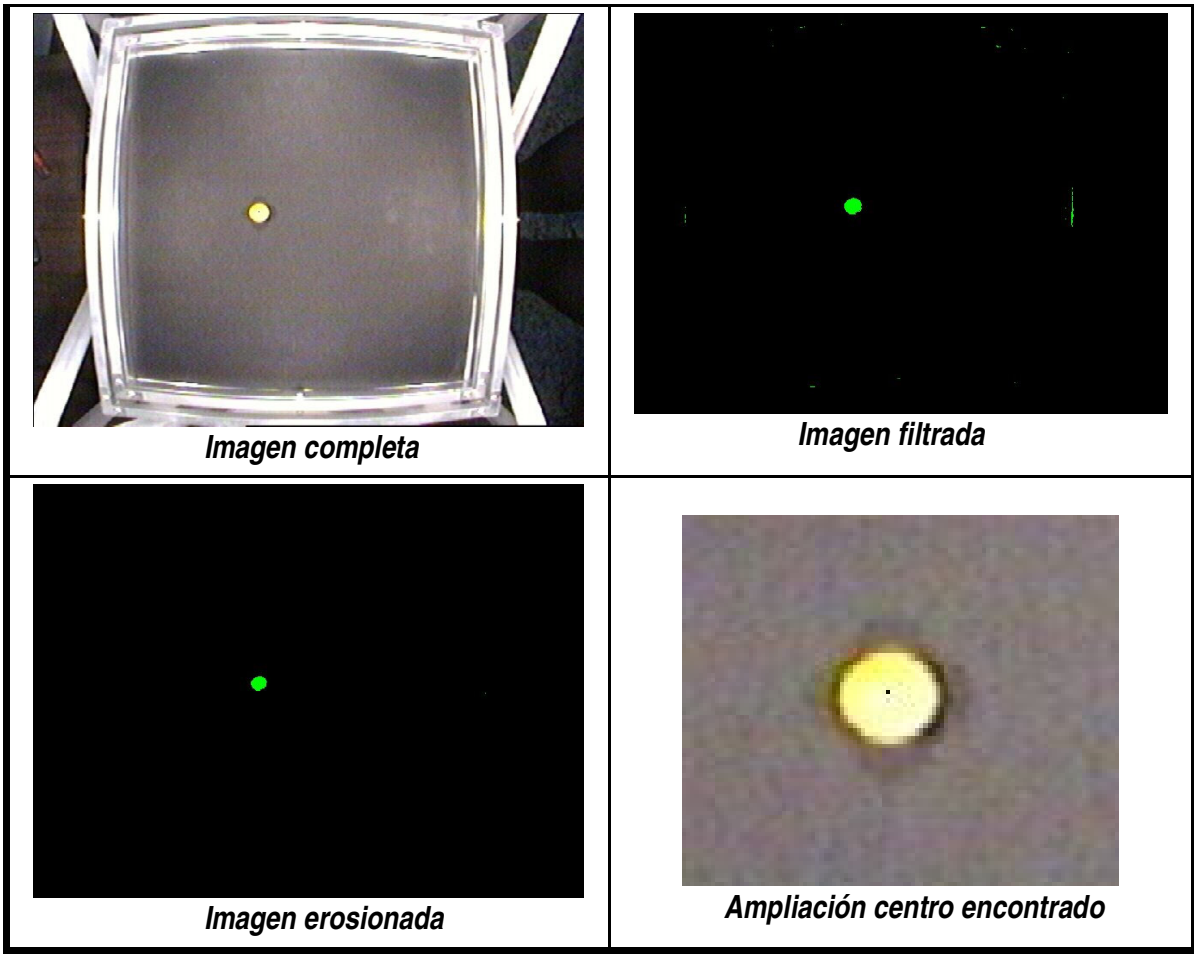


Fig 5.23 Etapas del proceso en imagen real

Podemos decir que:

- La iluminación en algunas zonas de la plataforma es excesiva, en algunos puntos aparecen reflejos codificados como blanco, lo cual dificulta el proceso de filtrado, ya que estas zonas se confunden con el reflejo de la estructura metálica.
- Exceptuando estos bordes metálicos, el proceso de filtrado elimina el 99.89 % de los puntos externos, y el erosionado todos.
- En el caso de procesar únicamente el entorno de la última posición, los resultados mejoran considerablemente, cuando la ventana no contiene partes del marco metálico, se eliminan todos los puntos del fondo en el proceso de filtrado.

Luego, colocando un regulador de luz, para controlar la luminosidad, podemos eliminar los brillos y obtener un resultado muy preciso.

En este punto, se consideró la opción de procesar sólo la parte de la imagen interior a los marcos, que en realidad es la única zona que puede contener a la bola. Sin embargo, esta opción

no es funcional, ya que al inclinarse las plataformas, el área varía. Para nuestra aplicación, es mejor llevar a cabo un buen procesado de la imagen en el que se eliminen estos puntos conflictivos, de este modo, además, la aplicación es reutilizable.

Para establecer una medida de la bondad de nuestro método, debido a que la calidad de estas imágenes es pobre, podemos delimitar los extremos de la bola y definir el centro verdadero como el punto medio de estos extremos:

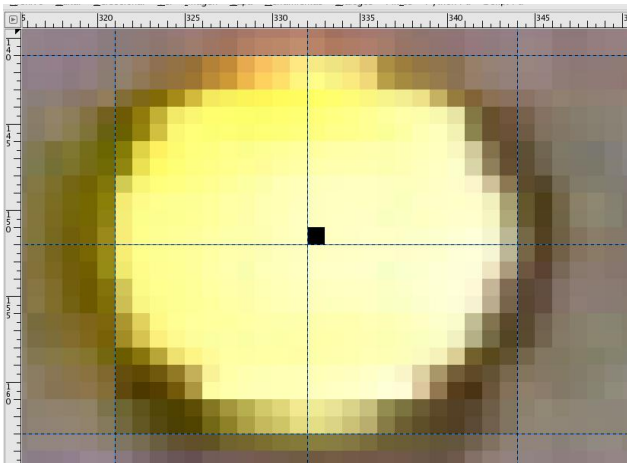


Fig 5.24 Error en el cálculo del centro

Como podemos ver en la imagen obtenida con la herramienta Gimp, en la que los ejes corresponden a pixeles, el error cometido es de un píxel en ambos ejes. Teniendo en cuenta los errores de píxelado (ya que en el cálculo del centro se redondea el centro a valor entero), podemos decir que el error es menor de un píxel en cada eje.

Vemos en la Tabla 5.3.a los resultados analizando una ventana de la imagen según la última posición, y también diezmando el número de pixeles de la imagen:

Procesado Completo	***Mínimo***	***Media***	***Máximo***
Carga_imagen	*47.26*	*47.31*	*47.46*
Filtrado	*9.98*	*10.56*	*12.24*
Erosionado	*35.21*	*37.23*	*44.21*
Búsqueda_Centro	*8.85*	*9.64*	*10.14*
Proceso completo	***101.3***	***104.6***	***114.05***

Tabla 5.3.a Tiempos de ejecución imagen no sobrecargada

Como hemos comentado, el tiempo de carga de la imagen a través de la librería Gandalf nos es indiferente ya que en nuestra aplicación final no se utilizará esta función.

Procesado de Imagen diezmada	***Mínimo***	***Media***	***Máximo***
Carga_imagen	*47.26*	*47.31*	*47.46*
Filtrado	*3.36*	*3.47*	*3.58*
Erosionado	*9.69*	*9.74*	*10.02*
Búsqueda_Centro	*3.54*	*3.62*	*3.81*
Proceso completo	***63.85***	***64.14***	***65.87***

Tabla 5.3 b. Tiempos de ejecución imagen diezmada no sobrecargada

Procesado de Ventana	***Mínimo***	***Media***	***Máximo***
Carga_imagen	*47.26*	*47.31*	*47.46*
Recortado_Ventana	*0.52*	*0.6*	*0.68*
Filtrado	*0.41*	*0.44*	*0.56*
Erosionado	*1.41*	*1.45*	*1.63*
Búsqueda_Centro	*0.18*	*0.2*	*0.26*
Proceso completo	***49.23***	***49.75***	***50.59***

Tabla 5.3 c. Tiempos de ejecución imagen enventanada no sobrecargada

Si no tenemos en cuenta el tiempo de cargar la imagen a través de la librería Gandalf:

Procesado:	***Mínimo***	***Media***	***Máximo***
Imagen completa de 640x480 pixeles	*45.63*	*57.29*	*64.66*
Imagen de 320x240	*15.94*	*16.68*	*18.59*
Ventana de 100x100 pixeles	*2.19*	*2.44*	*3.21*

Tabla 5.3 b. Tiempos de ejecución imagen no sobrecargada

Estos resultados son parecidos a lo que encontraremos al medir en MaRTE, algo mayores ya que hay que recordar que están medidos en Linux, con procesos del sistema ejecutándose simultáneamente. Sin embargo, estas medidas, confirman que, si bien los margenes temporales se cumplen ampliamente para el procesado de ventanas de 100x100 pixeles, procesar la imagen completa no es asumible más que en los casos en que sea imprescindible, como ya dijimos, en la primera iteración y cuando ocurran errores (pérdida de la bola en la ventana, o problemas del resto de las rutinas de control).

Por otra parte, la ventana de 100x100 pixeles estaba pensada en un principio para imágenes de 640x480. Ahora, para que la bola abandone el área de visión, debe recorrer 50 pixeles en 20 ms, es decir, la bola debe alcanzar algo más de 300 cm/sg para salir de la ventana, lo que supone unas 10 veces la velocidad máxima que la bola puede, físicamente alcanzar en el plano, calculada en el apartado 5.3.2.1. Luego, podríamos reducir las dimensiones de esta ventana de visión si fuese necesario un procesado más rápido.

6. INTEGRACIÓN DEL CONJUNTO:

6.1 APLICACIÓN PARA MARTE:

6.1.1 TIPOS DE DATOS:

Las funciones de la versión para MaRTE de la librería manejan los siguientes tipos de datos:

- Tipo en el que se guardan las coordenadas de un píxel desde la esquina superior izquierda:

```
struct xy{
    int     x;
    int     y;
  };
```

- Tipo en el que se guarda el contenido de color de cada canal de un píxel:

```
struct punto{
    int     r;
    int     g;
    int     b;
  };
```

- Umbrales máximo y mínimo de la función de umbralización:

```
struct umbral_RGB{
    struct      punto_min;
    struct      punto_max;
  };
```

- Tipo para almacenar imágenes:

```
typedef struct{
    unsigned char       b;
    unsigned char       g;
    unsigned char       r;
    unsigned char       alpha;
  }matriz[240][320];
```

Este tipo se utiliza para acceder al buffer de datos que recibimos de bttv_marte. Los tipos *unsigned char*, ocupan un Byte, que es precisamente el tamaño de cada campo de color de un píxel en el buffer. De este modo, en cada elemento de la matriz, se guardarán los cuatro valores del píxel.

6.1.2 INTEGRACIÓN CON LA LIBRERÍA BTTV_MARTE:

Utilizaremos las funciones descritas en el apartado 3.3 de esta librería, para acceder a las imágenes de la grabadora.

Para integrar nuestra librería de tratamiento de imagen con la librería bttv_marte, generamos dos nuevas funciones:

- **Para inicializar la cámara**

  *int **inicializa_camara** (void);*

Esta función inicializa la capturadora para que capture imágenes de 320x240 pixeles, con barrido no entrelazado, en color RGB con una profundidad de 4 Bytes, llamando a su vez a las funciones init_video_multibuffer y start_frame_grabber de bttv_marte La elección de estos parámetros se describe en el apartado 3.3.1.

Para asegurar el correcto funcionamiento de la cámara, la función incluye un retardo de tres segundos, ya que las primeras imágenes devueltas no son del todo estables.

Devuelve un entero de valor cero si se ejecuta sin errores, y menor que cero en caso contrario, elevando los códigos de error de las funciones init_video_multibuffer y start_frame_grabber de bttv_marte utilizadas.

- **Para obtener nuevas imágenes de la capturadora:**

```
int dame_foto ( foto            *matriz,
                struct timespec *instante);
```

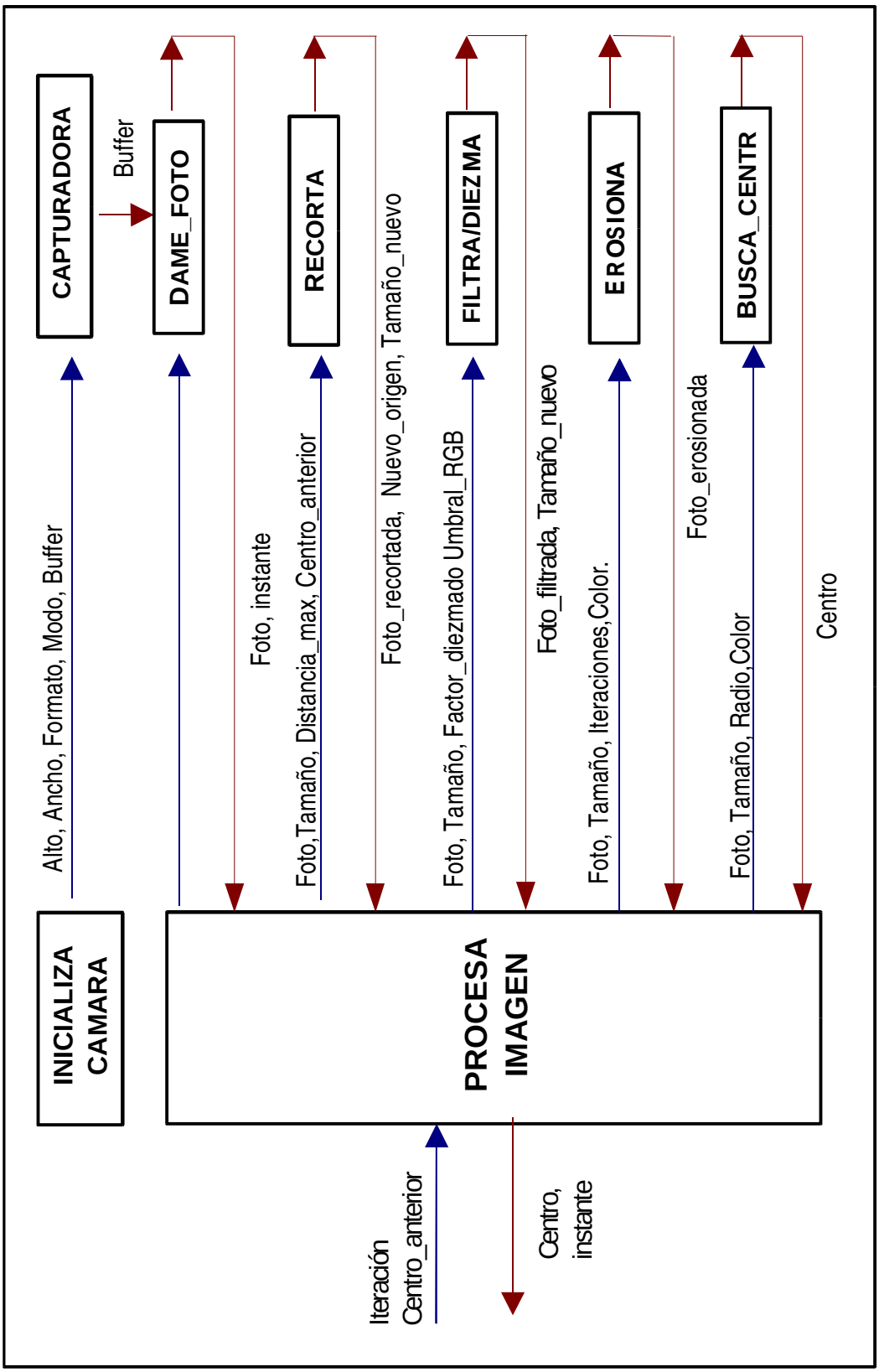
Devuelve en *foto* una copia del buffer interno que utiliza la capturadora, de este modo podemos trabajar con la imagen sin que esta sea sobreescrita. En instante se devuelve el instante de tiempo en nanosegundos (referido a la *época*)en que la imagen fue tomada.

Esta función es bloqueante, ya que realiza una llamada interna a la función *wait_for_next_image* de bttv_marte, como describimos en el apartado 3.3.1.

Devuelve un entero de valor cero si se ejecuta sin errores, y menor que cero en caso contrario, elevando los códigos de error de la función *wait_for_next_image* de bttv_marte.

6.1.3 PROCESOS:

La interfaz completa de la librería de imagen se describe en la cabecera *procesa_imagen.h.* La figura siguiente, muestra el intercambio de datos entre todas las funciones de la librería:


CAPTURADORA
Buffer
DAME_FOTO
RECORTA
FILTRA/DIEZMA
EROSIONA
BUSCA_CENTR
Alto, Ancho, Formato, Modo, Buffer
Foto, instante
Foto,Tamaño, Distancia_max, Centro_anterior
Foto_recortada, Nuevo_origen, Tamaño_nuevo
Foto, Tamaño, Factor_diezmado Umbral_RGB
Foto_filtrada, Tamaño_nuevo
Foto, Tamaño, Iteraciones,Color.
Foto_erosionada
Foto, Tamaño, Radio,Color
Centro
INICIALIZA CAMARA
PROCESA IMAGEN
Iteración
Centro_anterior
Centro, instante

6.1.4 TIEMPOS DE EJECUCIÓN EN MARTE OS:

Los tiempos medidos en MaRTE SO, confirman los resultados obtenidos en Linux. Vemos en la tabla 5.5, que son algo menores y podemos concluir que, necesitaremos una etapa inicial mayor que un período para procesar toda la imagen (ya que hay que tener en cuenta también el retardo del inicio de la capturadora), sin embargo a partir de ese momento, procesar ventanas de 100x100 pixeles puede hacerse en el intervalo requerido.

Procesado de imagen:	*Mínimo*	*Media*	*Máximo*
Imagen de 320x240	*7.313*	*8.233*	*17.226*
Ventana de 100x100 pixeles	*1.28*	*1.61*	*1.95*

Tabla 5.5 Tiempos procesado de imagen en MaRTE

Conviene recordar en este punto, que la función Dame_foto, no se ha tenido en cuenta hasta este momento, ya que al esperar a la siguiente imagen disponible en la capturadora, adquiere carácter bloqueante. Es decir, en el peor de los casos, nos hallaríamos en la situación de pedir una nueva imagen en el instante después de que la capturadora haya guardado una, sería necesario esperar entonces casi 20 ms.

Sin embargo es de esperar, que, si el resto de procesos se ejecuta en menos de 20 ms, a partir del momento en que se obtiene una imagen, el conjunto se sincronice, como muestra la figura 5.26:

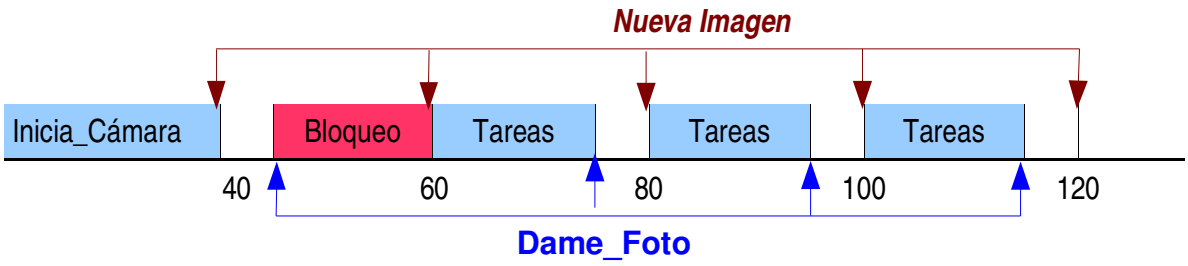


Fig 5.26 Sincronización con la captura de imágenes

Después de inicializar la grabadora, pedimos una nueva imagen, en este momento será necesario esperar un tiempo aleatorio entre 0 y 20 ms, en función del tiempo que falte para la próxima nueva imagen. El resto de procesos se encuentra bloqueado en este punto a la espera de

esta imagen. Una vez obtenida, se ejecutan las tareas de procesado de imagen y control de motores, que, como veremos en el capítulo siguiente, necesitan menos de 20 ms, a continuación se pide una nueva imagen, que tardará en obtenerse lo que queda de periodo. Es decir, no se producen más casos de bloqueo como consecuencia del carácter bloqueante de la función que carga las imágenes, por el contrario, esta sirve para sincronizar el resto de procesos.

6.2 PROCESOS DEL SISTEMA:

Como muestra la figura 6.1, el proceso completo se compone de varios subprocesos, sincronizados por la tarea Planificador.

- **Planificador:**

La tarea Planificador, en primer lugar inicializa los temporizadores que llevarán la cuenta de la duración del los periodos. A continuación, inicializa la capturadora mediante una llamada a la función *Inicializa_camara* (descrita en el apartado 5.5.1).

El siguiente paso es cargar la trayectoria deseada, que seguirá la bola a lo largo del plano, mediante una llamada a la tarea *Crea_Trayectoria.*

A partir de este momento, el planificador entra en un ciclo encabezado por la llamada fase inicial, a la que se volverá siempre que ocurra algún error que impida continuar alguna de las operaciones y que consiste en recolocar el plano en posición horizontal (llevado a cabo por la tarea *Control_motores*) y realizar el procesado de la imagen completa (llevado a cabo por *Procesa_imagen*).

Con este primer centro obtenido, el planificador llama a *Siguiente_Posición*, para obtener el siguiente punto de la trayectoria. En base a la posición actual de la bola, y la siguiente posición deseada, la tarea Control resuelve los ángulos necesarios en los motores para conseguir tal desplazamiento. Los ángulos, junto con el centro y velocidad de la bola (si ha sido encontrada) se escriben en un objeto protegido para ser leídos por la tarea Control_Motores, que se encargará de colocar los motores en la posición deseada (o en el estado inicial en caso de algún error).

- **Crea_Trayectoria:**

En nuestro caso, existen dos posibles trayectorias, una circunferencia alrededor del centro del tablero, la otra trayectoria trata de llevar la bola a un punto concreto.

Para calcular los puntos de la trayectoria, se establece una velocidad deseada, de manera que dos puntos de la trayectoria estén separados una distancia tal, que en un periodo se recorra en esa velocidad deseada.

■ Siguiente_Posición:

Este punto es calculado a partir de la posición y la velocidad actual de la bola según dos situaciones:

- Cuando la bola se encuentra cerca de la trayectoria, la siguiente posición devuelta será un punto de la trayectoria. Entendiendo por cerca, que será capaz de alcanzar algún punto de la trayectoria en un período a la velocidad deseada. La siguiente posición será el punto más cercano de la trayectoria a la posición actual de la bola.
- Cuando la bola se encuentra lejos de la trayectoria (no será capaz de alcanzar ningún punto de la misma en un periodo a la velocidad deseada), el siguiente punto estará a una distancia tal, entre el punto más cercano de la trayectoria y el actual, que la bola la recorra en un período a la velocidad deseada. Este punto se corrige teniendo en cuenta la dirección que lleva la bola, para conseguir que la bola alcance a la trayectoria de forma tangencial, de este modo, la bola no atravesará perpendicularmente la trayectoria una vez la alcance.

■ Control:

Calcula el ángulo que deben alcanzar los motores a partir de la posición actual y la deseada de la bola. Este proceso se lleva a cabo utilizando un controlador PID, en el que las partes derivativa y proporcional se mantienen constantes en todo el plano, mientras que el control integrativo se aplica sólo en los momentos en los que la bola se detiene, bien debido al rozamiento estático, o bien a que los ángulos de los motores son demasiado pequeños.

■ Control_Motores:

A partir de los ángulos deseados, esta tarea envía dos pulsos a la línea de datos del puerto paralelo de la plataforma de ejecución. La forma de hallar la anchura de los pulsos se describe en el apartado 3.1.2.

La figura 6.1 muestra un esquema del sistema completo y todos sus procesos:

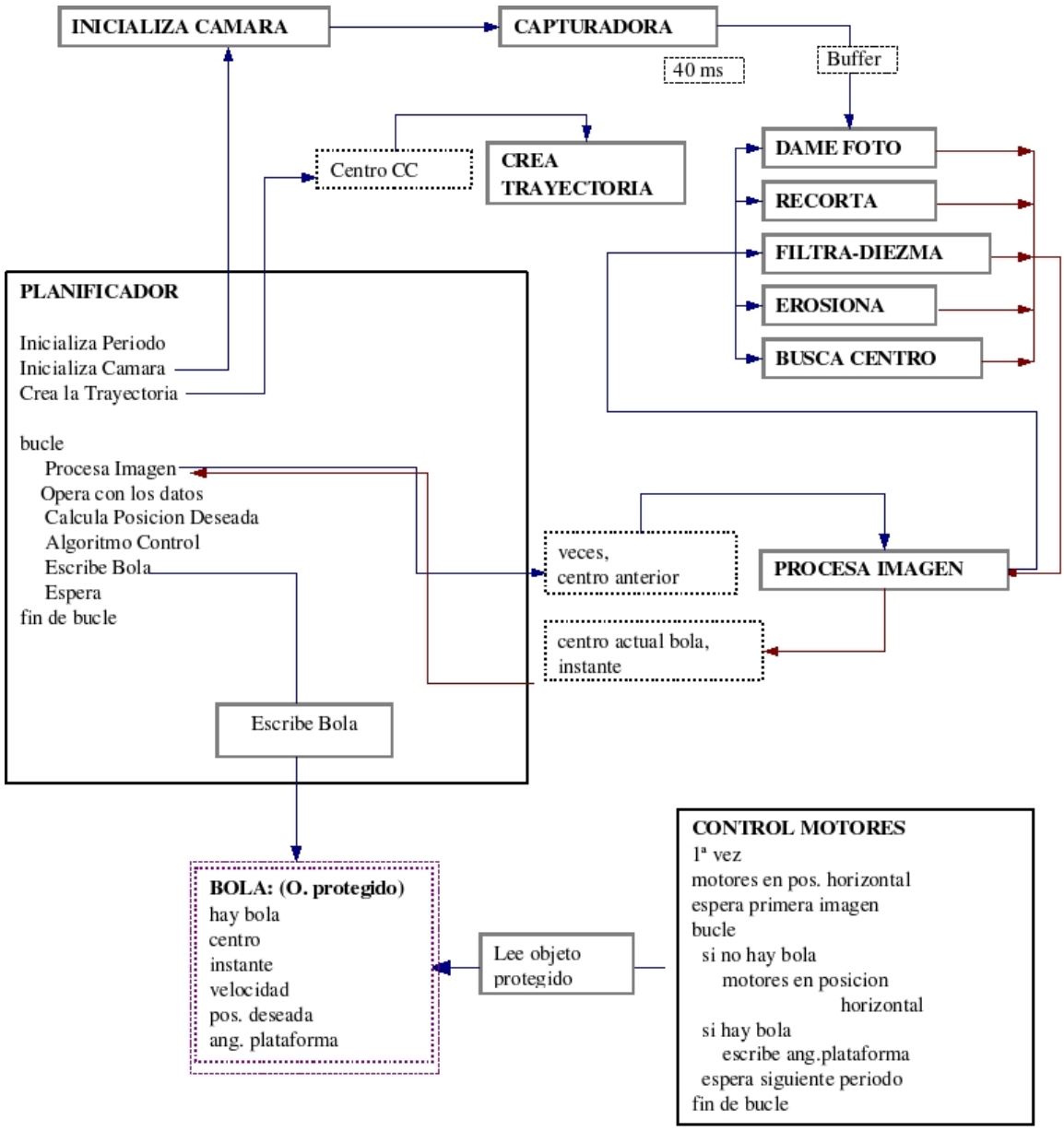


Fig 6.1 Procesos del sistema conjunto

6.3 PRIORIDADES Y PLAZOS DE EJECUCIÓN:

Para un correcto funcionamiento del sistema, se decidió asignar prioridades a las distintas tareas del sistema:

- La mayor prioridad se asigna a la tarea Control_Motores, ya que, como se describe en el apartado 3.1.2, los servomotores mantienen la inclinación siempre y cuando reciban señal cada 20 ms.
- El planificador, tiene menor prioridad y es el que se encarga de organizar la ejecución del resto de tareas.

Una vez implementado el sistema, el procesado de imágenes no entrelazadas presenta un gran inconveniente, el centro calculado a partir de una imagen correspondiente a una captura del campo par, difiere en un píxel en el eje *y* del centro calculado a partir de una captura del campo impar.

Cuando la bola está quieta en un punto del plano, el procesado de las imágenes devuelve dos valores distintos para la posición del centro, cada 20 ms, correspondientes a la variación del campo obtenido, con lo que el algoritmo de control entiende que la bola se está moviendo.

Hay que recordar, que la resolución de las imágenes es de 320 x 240 pixeles, correspondiendo los 240 a unos 30 cm, es decir, el error de un píxel, supone, para el algoritmo de control, que la bola se está moviendo a una velocidad de:

*1 píxel / 20msg * 30cm / 240pix = 6.25 cm/sg*

Por este motivo, se decidió procesar únicamente uno de los campos, cada 40 ms. No supone ningún problema para nuestros requerimientos temporales, puesto que ahora tenemos el doble de tiempo para ejecutar las tareas. Sin embargo, conviene recordar, que se necesita escribir cada 20 ms en los motores para que éstos mantengan la posición, por lo que las tareas de escritura en los motores llevarán un período de 20 ms.

Los tiempos de ejecución medidos en el sistema:

Proceso:	***Mínimo (ms)***	***Media (ms)***	***Máximo (ms)***
Inicializa_camara	*3.4897 (sg)*	*3.4898 (sg)*	*3.4999(sg)*

Proceso:		Mínimo (ms)	Media (ms)	Máximo (ms)
Crea_Trayectoria		*1.389*	*1.4439*	*1.459*
Procesa_imagen	100x100 pixeles	*1.281*	*1.62*	*1.95*
	320x240 pixeles	*7.313*	*8.2331*	*17.226*
	Esperando siguiente imagen (100x100)	*4.706*	*39.601*	*40.141*
	Esperando siguiente imagen (320x240)	*12.111*	*39.791*	*41.798*
Control		*0.029*	*0.02479*	*0.016*
Siguiente_Posicion		*9 *10^{-3}*	*14.59*10^{-3}*	*21 *10^{-3}*
Control_motores	Primera iteración	*0.5*	*0.5*	*0.5*
	En el ciclo	*1.271*	*1.587*	*1.768*

Tabla 6.1 Tiempos de ejecución del sistema completo

La figura 6.2 representa la ejecución del inicializado de la Cámara, con las tareas Planificador y Control_Motores ejecutándose concurrentemente, suponiendo los máximos retardos. El planificador inicia la capturadora, carga la trayectoria y procesa una primera imagen completa, mientras, el control de motores coloca los servos en la posición de partida en un máximo de 0.5s. Se espera 0.5 segundos, porque es el tiempo de respuesta máximo de los motores (de la posición máxima a la mínima o viceversa).

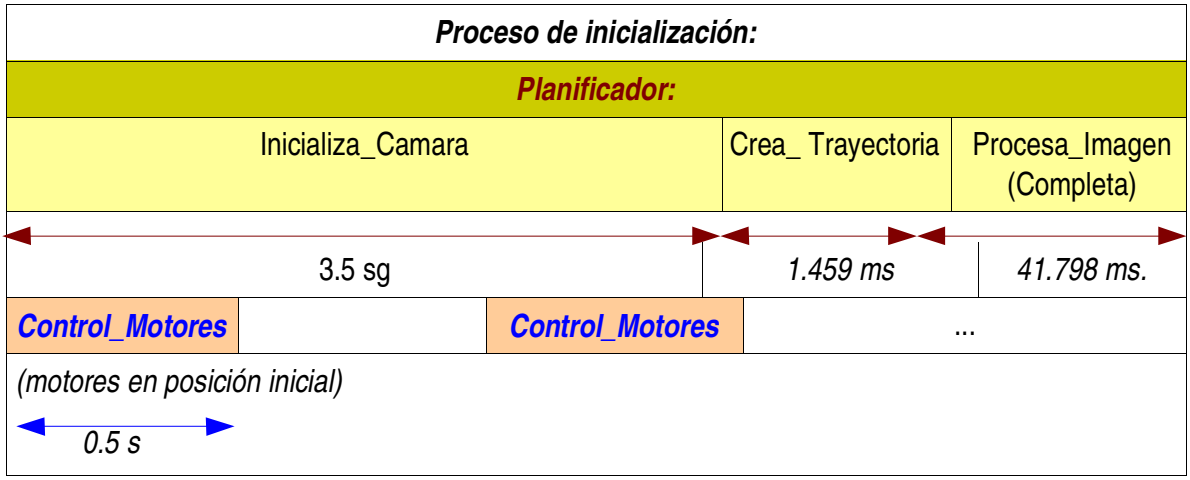


Fig 6.2 Proceso de inicialización del sistema

La figura 6.3, representa la ejecución de un periodo del ciclo, suponiendo nuevamente los retardos máximos. PROCESA_IMAGEN pide dos imágenes consecutivamente, de las cuales descarta la primera, de ese modo siempre opera con el mismo campo (el par o el impar).

Cuando se ejecute por primera vez, la primera llamada a Dame_Foto, se necesitará un tiempo aleatorio entre 0 y 20 ms para obtener la siguiente imagen, mientras que la segunda llamada, necesitará forzosamente 20 ms (al haber proporcionado inmediatamente antes la nueva imagen). Estos 20 ms son los que hemos decidido esperar, para obtener el mismo campo que en la iteración anterior. Es decir, ahora en la primera iteración de Dame_Foto, se espera un tiempo aleatorio entre 20 y 40 ms, en las siguientes iteraciones los procesos se sincronizan como vimos en la figura 5.26.

A partir de esta primera imagen, Procesa_Imagen calcula el centro de la bola, y lo pasa al Planificador, que hasta este momento se mantenía a la espera del nuevo dato. Se resuelven las posiciones de los motores y se mueven éstos. A continuación, el planificador solicita una nueva imagen.

Desde que se obtiene esta imagen hasta que el Planificador termina sus tareas, pasará un tiempo menor que la suma de los máximos tiempos de ejecución del resto (Cálculo del centro, Siguiente_Posición y Control). Por lo tanto, en las siguientes iteraciones del ciclo, la primera imagen válida tardará en obtenerse un máximo de 38.034 ms, a partir de este momento las tareas quedan sincronizadas y se ejecutan dentro de los plazos previstos.

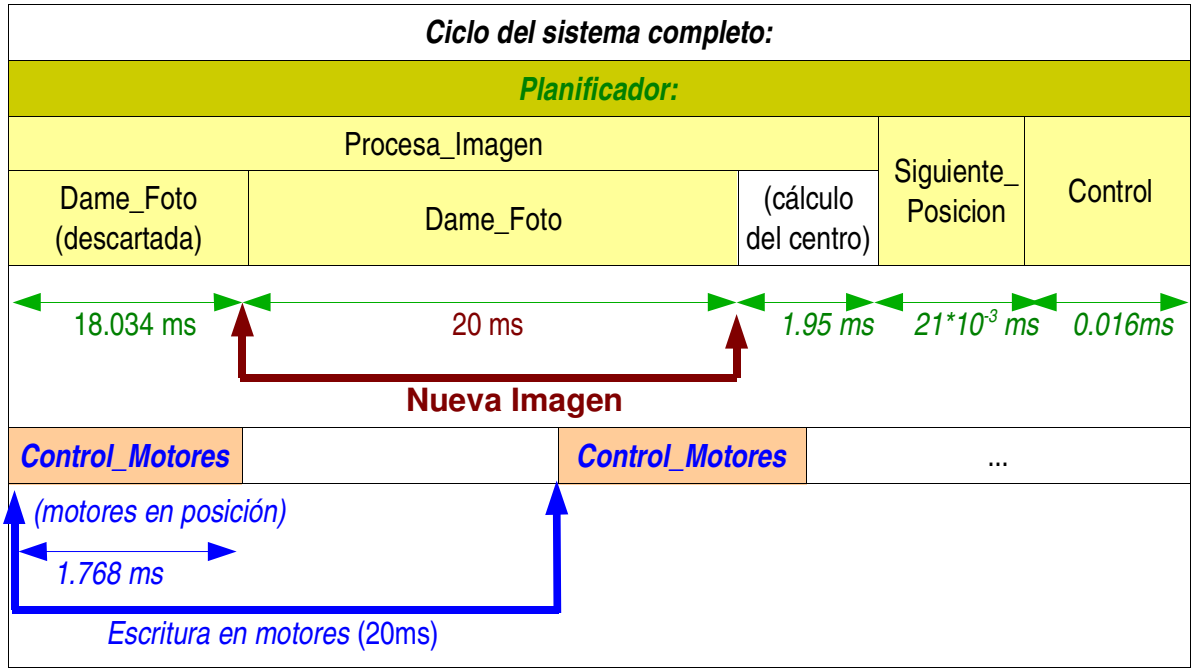
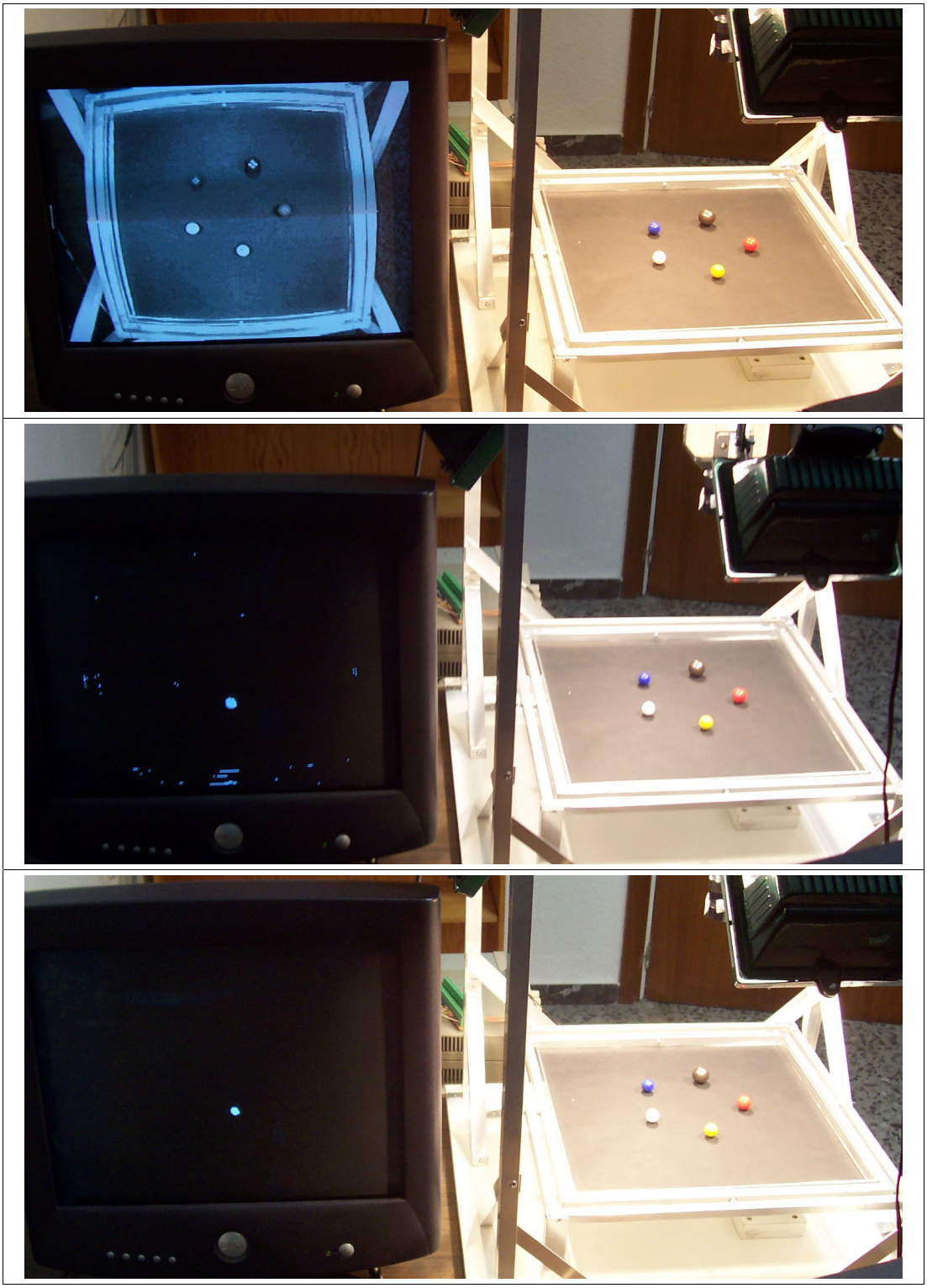


Fig. 6.3 Ciclo del sistema completo

Es interesante destacar, que de los 40 ms del ciclo, solo unos 4 ms se destinan al procesado de imagen y control de los motores, el resto del tiempo, se espera.

La siguiente figura muestra un ejemplo de las distintas etapas del procesado de imagen, bajo MaRTE OS:



En la primera imagen, se muestra por pantalla la imagen leída del buffer. A continuación, tenemos el resultado de la etapa de filtrado, como vemos, cuatro de las bolas han desaparecido, quedando sólo el brillo de dos de ellas. Como puntos externos permanecen solamente puntos del marco de las estructura metálica.

Como ya vimos al procesar imágenes en Linux, la etapa de filtrado no eliminará estos brillos por que el objeto que buscamos también los tiene y no nos conviene un filtrado agresivo. En las tres escenas, observamos a la izquierda el monitor de la plataforma de ejecución y a la derecha el tablero, con cinco bolas de distintos colores.

Comprobamos en la última imagen, la etapa de erosionado elimina *todos* los puntos externos, permitiéndonos obtener la posición del centro de la bola.

Es importante señalar, que en la aplicación final, las etapas intermedias no se presentarán por pantalla, ya que consumiría recursos y estaríamos ante un proceso cuyo tiempo de respuesta no sería acotado.

7. CONCLUSIONES Y TRABAJO FUTURO:

CONCLUSIONES:

Se ha cumplido el objetivo principal del proyecto descrito en el apartado 1.2, de estudiar el comportamiento de MaRTE OS en una aplicación de tiempo real dotada de visión, y se ha comprobado que el sistema conjunto satisface los requerimientos temporales establecidos.

Se ha obtenido una librería de tratamiento de imagen capaz de cumplir los requerimientos temporales del sistema, con versiones para Linux como para MaRTE, con unos resultados que cumplen los objetivos previstos en un principio.

Para el procesado de imágenes, esta librería incluye funciones que permiten:

- Entrada y salida de imágenes (a través de la librería Gandalf en Linux, y a través de Bttv_Marte, en MaRTE).
- Diezmado de imágenes (mediante muestreo, obteniéndose imágenes de la resolución deseada).
- Enventanado de imágenes (para obtener una nueva imagen de dimensiones deseadas a partir de un punto dado de la imagen).
- Umbralizar y binarizar imágenes con umbrales variables (para distinguir los puntos que pertenecen al objeto del fondo, según los umbrales RGB deseados).
- Calcular el histograma de imágenes RGB y escala de grises (y hallar así los umbrales de binarización).
- Erosionar imágenes (para eliminar puntos aislados externos al objeto, tanto para imágenes RGB como en escala de grises y el número de iteraciones deseado).
- Corrección de la distorsión radial introducida por la cámara analógica.

Los resultados dependen en gran medida de los umbrales RGB establecidos en la etapa de umbralizado. Así con unos umbrales muy estrictos, se obtendrá un umbralizado 'agresivo'. Se eliminarán la mayoría de los puntos externos y como consecuencia, las iteraciones de erosionado

necesarias serán mínimas. La etapa de erosionado es la que más tiempo consume, por lo que se obtendrán resultados en poco tiempo. Sin embargo, el umbralizado agresivo puede eliminar puntos del objeto, por lo que, estos resultados pueden no ser exactos.

Si optamos por umbrales RGB menos estrictos, conservaremos todos los puntos del objeto, pero serán necesarias varias etapas de erosionado para eliminar el resto de puntos. Es necesario por tanto, llegar a un compromiso entre tiempo de procesado y exactitud de los resultados.

En nuestra aplicación, la solución de enventanar en un entorno de la última posición conocida de la bola, permite obtener resultados precisos y cumplir las restricciones temporales.

La librería ha sido integrada con éxito con las tareas de control de motores y con la librería de captura de imágenes en MaRTE, incluyendo funciones en la librería para inicializar la capturadora, leer imágenes del buffer de datos y dibujar imágenes en pantalla, consiguiéndose el correcto intercambio de datos entre todas las partes del sistema y dentro de los plazos establecidos.

El uso de la cámara analógica, dadas sus características supone una de las mayores limitaciones del sistema. Por un lado el barrido entrelazado nos obliga a reducir la resolución prevista en un principio (640x480 pixeles) haciéndonos perder calidad de imagen y por tanto introducir error adicional en el cálculo de la posición del centro de masa. Por otra parte, la distorsión introducida por el desfase de un píxel entre los campos par e impar nos obliga a procesar la mitad de imágenes, de los 40 ms de cada período, un 90% se destina a esperar una imagen válida.

Sin embargo, a pesar de las limitaciones descritas, el sistema encuentra el centro de masas de objetos de color amarillo en un tiempo relativamente bueno. Aunque la aplicación sólo incluye una bola, se hicieron pruebas con bolas de distintos colores y el sistema demostró ser capaz de distinguir el objeto de interés. Además, incluso a velocidades muy altas (incluso mayores que la máxima que la bola alcanza por si sola en el plano), la bola nunca desaparece de la ventana de visión. Ocultando manualmente la bola, se comprueba asimismo que el sistema es capaz de situar los planos en posición horizontal y mantenerse a la espera.

La trayectoria pensada en un principio, que la bola recorriese una trayectoria circular en el plano, fue sustituida por una más sencilla, ya que, la resolución de la cámara, el salto de un píxel entre los campos par e impar y las características del algoritmo de control no permitieron conseguir resultados satisfactorios. Sin embargo, el sistema conjunto, es capaz de llevar la bola a cualquier punto deseado del plano, controlarla, y hacer que se quede quieta en ese punto. Por lo tanto, el objetivo principal del sistema conjunto, controlar la bola, se ha logrado para trayectorias sencillas.

TRABAJO FUTURO:

El siguiente paso lógico, sería sustituir la cámara analógica por una digital, para la que se están desarrollando drivers en este momento. Esta modificación, permitiría obtener imágenes a mayor frecuencia, y sin el inconveniente del barrido entrelazado de la cámara analógica. Como se ha visto, procesar imágenes de mayor resolución no supondrá ningún inconveniente, ya que las tareas de procesado de imagen y control de los motores se ejecutan en un plazo mucho menor que el periodo.

Por otro lado, el sistema de captura de imágenes es demasiado dependiente de la cantidad de luz incidente, este hecho, sumado a la mala calidad del obturador de la cámara, hacen muy difícil el proceso de búsqueda de los umbrales de binarización de la imagen, que es en esencia el único parámetro que habría que modificar para que el sistema encontrase otro tipo de objetos. Sería interesante colocar una cámara con iris motorizado de forma que la exposición de ésta pueda ser controlada y ajustada por software, o bien una una cámara equipada con Auto Iris que intentará producir una señal de vídeo de brillo constante y abrirá o cerrará el iris de las lentes, conforme varíen los niveles de luz.

BIBLIOGRAFÍA:

[ADWE1996] Addison-Wesley "Digital image processing", 1996

[ALDM2002] Aldea Rivas Mario, "Planificación de tareas en sistemas sistemas operativos de tiempo real estricto para aplicaciones empotradas", 2002.

[DELA2001] De la escalera, Arturo " Visión por computador. Fundamentos y métodos", 2001

[BRBU2000] Bräuer - Burchardt, Ch. & Voss, K. (2000). "Automatic Lesns Distortion using single views " *2000 (Proc. of 11. GAGM Symposium, Kiel, 200)*

[ENAL2000] Enrique alegre " Procesamiento digital de imagen: fundamentos y métodos" 2003

[GONJ2000] Gonzalez Jimenez Javier "Vision por computador", 2000.

[GONM2001] Gonzalez Michael, "Posix de tiempo real" Universidad de Cantabria,2001

[PERZ2003] Pérez Vega Constantino, Zamanillo Sainz de la Maza José Mª "Fundamentos de televisión analógica y digital", 2003

[PETM1996] Petros Maragos, Ronald W. Schafer, Muhammad Akmal Butt "Mathematical morphology and its applications to imagen and signal processing", 1996

[REK1994] Reinhard Klette, Piero Zamperoni "Handbook of image processing operators",1994.

[RKTS1998] R. K. LENZ, R. Y. TSAI. Techniques for calibration of the scale factor and image center for high accuracy 3-D machine vision metrology. IEEE Transactions on PAMI, vol. 10, No. 5. Septiembre, 1988.

[SCH1989] Schalkoff, Robert J "Digital image processing and computer vision", 1989

[SIL1997] Silberschatz, Abraham, Baer Galvin Peter. "Sistemas operativos",1997

[TSA1987] Tsai, Roger Y., "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using o_-the-shelf TV ameras and lenses, " IEEE Journal of Robotics and Automation , vol. 3, No. 4, Agosto 1987.

[VOSS1995] Voss, K. y Suesse H. "Affine normalization of planar regios by moments using a new separation method".

[VOSS2001] Voss, K y Braüer, "Rectificación monocular de imágenes" Computación y sistemas, 2001

Enlaces de interés:

[1] http://marte.unican.es/

[2] http://gandalf-library.sourceforge.net/

[3] http://www.etsimo.uniovi.es/vision

- Formatos imagen y video:

[4] dfists.ua.es/tavarca/ponencias/geocolor.pdf

[5] fbio.uh.cu/ilene/ntic/conterencia_profe.pdf

- Morfología matemática:

[6] homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm

[7] www.dc.uba.ar/people/materias/ pdi2/morfologia_binaria.doc