BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

TRABAJO DE GRADO

Seguimiento de Trayectorias en Visión por Computadora

Nombre y Apellido: *María Carina Roldán*

Número de Alumno: *1228/7*

Facultad de Informática - U.N.L.P.

TES
00/12
DIF-02121
SALA

UNIVERSIDAD NACIONAL DE LA PLATA
FACULTAD DE INFORMATICA
Biblioteca
50 y 120 La Plata
catalogo.info.unlp.edu.ar
biblioteca@info.unlp.edu.ar

DIF-02121

2000

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

Tabla de Contenidos

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

Motivación y Objetivo General

En muchas aplicaciones se debe seguir la trayectoria de una entidad, característica u objeto sobre una secuencia de imágenes.
Cuando el procesamiento debe hacerse en tiempo real, lleva consigo una complejidad importante y requiere esfuerzo en la paralelización de los algoritmos.
En el presente proyecto, se estudiará el caso particular de objetos simples moviéndose en un contexto alcanzable por el elemento de visión (cámara de video). Serán los objetivos principales:

- Analizar y comprender las etapas relacionadas con los sistemas de visión por computadora, como son el mecanismo de tracking y el reconocimiento de objetos en una secuencia de imágenes.

- Desarrollar los algoritmos que implementen una solución para dicho problema, permitiendo extraer de una secuencia de imágenes la información de objetos y sus movimientos.

- Estudiar la paralelización de los algoritmos involucrados tanto en el proceso de reconocimiento de objetos como en el de seguimiento de sus trayectorias.

- Implementar esta paralelización por medio de una herramienta de programación adecuada.

- Por último, realizar un análisis cuantitativo del comportamiento de los distintos algoritmos y evaluar los resultados obtenidos. Esta evaluación permitirá conocer la efectividad de la paralelización realizada, y determinar bajo qué condiciones resulta ser ésta la mejor solución.

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

Descripción del Problema

El seguimiento de los objetos en una secuencia de imágenes, como se dijo está particularmente asociado con:

- El reconocimiento y clasificación de objetos.
- El seguimiento de las trayectorias de estos objetos a través de esa secuencia.

Esta idea se puede ver con el siguiente diagrama:

Varios son los conceptos teóricos involucrados para el desarrollo de esta propuesta. Estos incluyen principalmente:

- Procesamiento de imágenes por computadora
- Seguimiento de trayectorias
- Arquitecturas y lenguajes de programación paralelo

Por lo tanto, antes de desarrollar el tema central de la propuesta, se introducirán todos estos conceptos. Se incluirá también una reseña sobre MPI – Message Passing Interfase – que es el paradigma de programación elegido para la implementación.

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

Procesamiento de Imágenes y Visión por Computadora

El objetivo de un sistema de visión por computadora es crear un modelo del mundo real a partir de imágenes. Existen diferentes tipos de aplicaciones de un sistema de visión por computadora entre los cuales se pueden mencionar el diagnóstico de enfermedades usando imágenes de tomografías computadas o el control de calidad de productos que van desde alimentos hasta piezas industriales. Otro tipo de aplicaciones tiene que ver con la recuperación de estructura tridimensional de un ambiente, es decir, el reconocimiento e interpretación de los movimientos en una escena; estas aplicaciones son útiles en la automatización de navegación de autos, aviones o robots. Un tercer grupo de aplicaciones dentro de la visión por computadora tiene que ver con el análisis y el manejo de volúmenes excesivamente grandes de datos obtenidos por satélites; estas imágenes son importantes en el pronóstico del tiempo, el análisis del comportamiento del planeta, la agricultura, etc.

Un campo estrictamente relacionado con la visión por computadora es el procesamiento de imágenes. El término *procesamiento digital de imágenes* se refiere generalmente a procesar una imagen bidimensional por medio de una computadora. Una imagen digital es un arreglo de números reales o complejos representados por un número finito de bits. Las formas en que se puede encontrar originalmente una imagen a procesar van desde un dibujo o fotografía hasta un cuadro de una cinta de video. Con el fin de poder analizarse, estas imágenes son digitalizadas y grabadas como una matriz de dígitos binarios en algún soporte de almacenamiento digital. La imagen digitalizada puede entonces ser procesada según lo requiera cada aplicación individual.

Las técnicas de procesamiento de imágenes usualmente transforman imágenes en otras imágenes. Este campo incluye tópicos tales como el realce de imágenes, la compresión de imágenes y la corrección de imágenes borrosas o fuera de foco. Por el contrario, los algoritmos de visión por computadora toman imágenes como entrada pero producen otro tipo de salida tal como la representación del contorno de un objeto encontrado en una escena. Los algoritmos de procesamiento de imágenes son útiles en las primeras etapas de un sistema de visión por computadora. Usualmente se utilizan para enfatizar una determinada información y para suprimir el ruido.

Realce de Imágenes

El realce de imágenes tal como se mencionó, tiene como objetivo acentuar ciertas características de las imágenes para un subsecuente análisis o despliegue de las mismas en un dispositivo de salida. Como ejemplos de esta operación se puede mencionar el realce del contraste, de los bordes, la reducción de ruidos, etc. El proceso de realzar o mejorar una imagen no aumenta la información de la misma; simplemente enfatiza ciertas características lo cual facilita el procesamiento que se le dé con posterioridad. El realce de imágenes es útil en prácticamente todas las aplicaciones de procesamiento de imágenes. A continuación se muestran algunas de las técnicas más comúnmente usadas:

Operaciones Puntuales

Las operaciones puntuales son operaciones donde un nivel de gris dado es mapeado a otro nivel de gris acorde a una función. Algunos ejemplos de operaciones puntuales son:

- Expansión del Contraste: Se realiza una expansión de los intervalos de la escala de grises donde los pixels son más frecuentes, de manera de lograr una mejora en la visibilidad de

una escena. La transformación a realizar puede obtenerse a partir de un examen del histograma de la imagen.

- Nivelación (thresholding): Es una transformación que lleva una imagen en escala de grises a una imagen binaria. Por ejemplo, una imagen correspondiente a una página impresa, al ser escaneada no generará una imagen binaria como se espera debido al ruido y/o variaciones en la iluminación. Para lograr la imagen binaria se puede utilizar thresholding.

Operaciones Espaciales

Las operaciones espaciales se realizan sobre los vecinos locales de los pixels de entrada. A menudo se realiza la convolución de la imagen con una máscara espacial. Esta idea se entenderá presentando algunos ejemplos de operaciones espaciales sobre imágenes:

- Promedio espacial: Cada pixel es reemplazado por un promedio pesado de sus pixels vecinos, esto es:

$$v(m,n) = \sum\sum_{(k,l)\in W} a(k,l)\, y(m-k,\, n-l)$$

donde $y(m,n)$ y $v(m,n)$ son las imágenes de entrada y salida respectivamente, W es una ventana elegida, y $a(k,l)$ son los pesos del filtro. Un caso especial de filtro de promedio espacial es el que tiene todos los pesos iguales, dando:

$$v(m,n) = \frac{1}{N_W} \sum\sum_{(k,l)\in W} y(m-k,\, n-l)$$

donde $a(k,l) = 1 / N_W$ y N_W es el número de pixels en la ventana W.

La siguiente figura muestra algunas máscaras de promedio espacial.

¼	¼
¼	¼

ventana de 2x2

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

ventana de 3x3

0	1/8	0
1/8	¼	1/8
0	1/8	0

promedio pesado de 5 puntos

El promedio espacial es utilizado para suavizar el ruido en una imagen. En la práctica el tamaño de la ventana W es limitado debido a que con el tamaño, el promedio espacial introduce una distorsión dejando la imagen borrosa.

- Filtro de la mediana: En este caso el pixel de entrada es reemplazado por la mediana de los pixels contenidos en una ventana alrededor del pixel. Esto es:

$$v(m,n) = mediana\{y(m-k,\, n-l), (k,\, l) \in W\}$$

donde W es la ventana elegida. El algoritmo para este filtro requiere acomodar los valores de los pixels de la ventana en orden creciente o decreciente y tomar el valor del medio.

Generalmente el tamaño de la ventana se elige de manera que el número de pixels es par; caso contrario se toma como mediana el promedio de los dos valores del medio.
Típicamente se escoge una ventana de 3x3, 5x5, 7x7 o la ventana de cinco puntos del ejemplo del promedio espacial.
El filtro de la mediana es útil para remover líneas o pixels aislados preservando la resolución espacial. Se comporta my bien para imágenes con ruido binario pero es muy pobre cuando el ruido es más desparejo, en cuyo caso es más adecuado un filtro de promedio espacial. Tampoco resulta bueno cuando el número de pixels con ruido en la ventana es mayor o igual que la mitad del número de pixels en dicha ventana.

El siguiente ejemplo ilustra este tipo de filtro usando una ventana de 3x3.

75	99	36
38	49	10
19	98	22

Ordenados por el valor del pixel:
10 – 19 – 22 – 36 – **38** – 49 – 75 – 98 – 99

El nuevo valor del pixel es **38**.

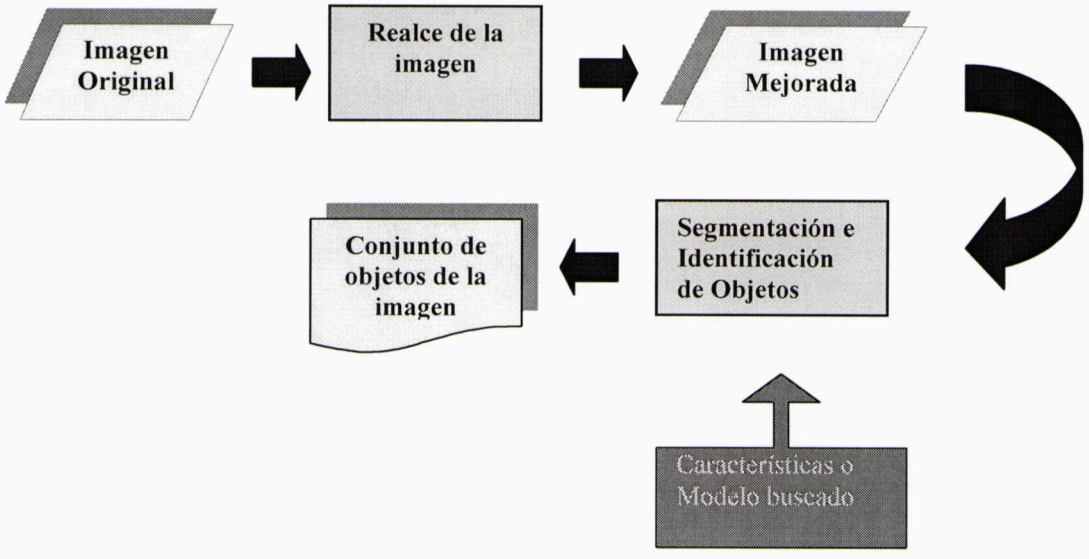
Modelización del Histograma
El histograma de una imagen representa la frecuencia relativa de la ocurrencia de los distintos niveles de gris en la imagen. Las técnicas de modelización del histograma modifican una imagen de modo que su histograma tenga una forma deseada. Esto es útil al expandir los niveles de bajo contraste de imágenes con histogramas estrechos. Una de estas técnicas es la ecualización, cuyo objetivo es obtener una imagen con un histograma uniforme.

Segmentación

La segmentación consiste en identificar los componentes semánticamente significativos de una imagen y agrupar los pixels que pertenecen a tales componentes. La segmentación no necesita realizarse en términos de objetos; pueden usarse también algunos predicados basados en características de intensidad. Estos predicados son usualmente llamados "características" o "*features*".
El procedimiento para realizar la segmentación se simplifica si se tienen imágenes claras, nítidas, por lo cual retocarlas previamente (mejorar el contraste, realzar el color, enfatizar los bordes, etc.) constituye un paso de gran ayuda.

Como resultado del proceso de segmentación se obtiene un conjunto de regiones u objetos. A cada uno de éstos se lo puede identificar por:

- Una ubicación, dada por un par de coordenadas (x,y) que lo sitúen espacialmente en el cuadro. Estas coordenadas podrían coincidir con el centro del objeto o región.
- Un conjunto de características: tamaño o área, intensidad promedio, orientación, perímetro, etc., según la relevancia que tengan para la aplicación que realice el proceso.



Existen varias técnicas de segmentación de imágenes; algunas de ellas son detalladas a continuación:

Nivelación de amplitudes (Thresholding)

La nivelación por amplitud es útil cuando las características de amplitud son suficientes para caracterizar al objeto. En este caso un intervalo de amplitudes dado representa una única característica de objeto. Esta técnica es útil por ejemplo en la segmentación de documentos impresos o imágenes de rayos X.

La selección del *threshold* o nivel de separación es un paso importante en este método. Algunas soluciones comúnmente usadas para esta selección son las siguientes:

1. Examinar el histograma buscando picos y desniveles. Luego estos desniveles pueden ser usados para la selección.
2. Seleccionar el *threshold* (t) de modo que una predeterminada fracción del número total de muestras esté exactamente por debajo de t.
3. Realizar una selección adaptable, examinando los histogramas parciales.

Etiquetado de Componentes (Labeling)

Es un método de segmentación de imágenes binarias simple y efectivo. Consiste en examinar la conectividad de pixels con sus vecinos y etiquetar o colocar *labels* a los conjuntos conectados. Un algoritmo posible para realizar este proceso es llamado *etiquetado de pixels*, y es como sigue:

Supóngase que una imagen binaria es escaneada de izquierda a derecha y de arriba hacia abajo. Tomando como muestra la siguiente figura, el pixel actual X es etiquetado como perteneciente o bien a un objeto (1) o a un agujero (0) examinando su conectividad a sus vecinos A, B, C y D.

C	A	D
B	**X**	

Por ejemplo, si **X** es 1, entonces es asignado al o los objetos a los cuales está conectado. Si hay dos o más objetos candidatos, entonces esos objetos son declarados equivalentes y son integrados en uno solo. Por ejemplo, sea el siguiente caso:

0	1	1
0	1	0
1	**X**	

0	3	3
0	3	0
5	?	

0	3	3
0	3	3
3	3	

donde el primer cuadro (a) muestra los valores de los pixels individuales y el segundo (b), las etiquetas asignadas. En este caso hay que asociarle una etiqueta a **X**. Como está conectado a dos pixels con etiquetas diferentes, 3 y 5, éstas se declaran equivalentes y la nueva asignación es como muestra el tercer cuadro (c).
Cuando se detecta una transición de un 0 a un 1 aislado, se asigna una nueva etiqueta o *label* de objeto. Una vez que el pixel es etiquetado, se actualizan las características del objeto al que pertenece.
Como resultado, al final del recorrido de la imagen se tienen para cada región de 1s conectados todas las características recopiladas, como área, perímetro, etc.

Correspondencia de patrones (Template Matching)
Un método directo para segmentar es comparar contra patrones de una lista dada. Los objetos detectados pueden ser segmentados y quitados de la imagen para luego analizar el resto de la imagen utilizando otras técnicas.

Reconocimiento de Objetos

Un sistema de reconocimiento de objetos encuentra objetos del mundo real a partir de una imagen capturada del mundo real, usando modelos de objetos conocidos a priori.
El problema de reconocimiento de objetos está estrechamente relacionado con el problema de segmentación. Sin al menos un reconocimiento parcial de objetos, no se puede segmentar y sin segmentación, el reconocimiento de objetos no es posible.
Algunos de los principales temas a considerar en el diseño de un sistema de reconocimiento de objetos son: ¿Cómo representar los objetos? ¿Cuáles son los atributos o características importantes que se deben detectar? ¿Cómo pueden compararse las características de una imagen con los modelos?

La complejidad del reconocimiento depende de varios factores. Por ejemplo, si las condiciones en las que se capturó una imagen (iluminación, fondo, punto de vista) no son similares a las condiciones bajo las cuales se definió el modelo, el problema se vuelve más complejo. Otro factor es la oclusión. Si hay sólo un objeto en la imagen, puede ser completamente visible. A medida que crece el número de objetos en la imagen, la posibilidad de oclusión aumenta. Esto en muchos casos representa un serio problema.

Muchos tipos de característica son usados para el reconocimiento de objetos. Algunas de las características comúnmente usadas son:

- Globales: Son características de regiones obtenidas usando todos los puntos del objeto, sus ubicaciones, intensidad, etc. Ejemplo de este tipo de características son el tamaño o perímetro.
- Locales: Se refieren al contorno de un objeto; representan una parte distinguible de una región, por ejemplo la curvatura o las esquinas.
- Relacionales: Se basan en las posiciones relativas de distintas características o regiones, por ejemplo distancia entre componentes u orientación.

Visión Dinámica

Si bien los primeros sistemas de visión por computadora estaban relacionados principalmente con escenas estáticas, para varias aplicaciones se han diseñado sistemas de visión por computadora que analizan escenas dinámicas. La entrada a un sistema de análisis de escenas dinámicas es una secuencia de cuadros de imágenes tomadas de una escena real con movimientos. Cada cuadro representa una imagen de la escena en un instante preciso en el tiempo. Los cambios en la escena pueden deberse al movimiento de la cámara, al movimiento de los objetos, a cambios en la iluminación, o a cambios en la forma o tamaño de los objetos. El sistema debe detectar cambios, determinar las características de los movimientos, recuperar la estructura de los objetos y reconocer los objetos que están en movimiento.

Al momento de analizar una escena dinámica, se tienen cuatro posibilidades:

1) Cámara estacionaria y objetos estacionarios (SCSO)
2) Cámara estacionaria y objetos en movimiento (SCMO)
3) Cámara en movimiento y objetos estacionarios (MCSO)
4) Cámara en movimiento y objetos en movimiento (MCMO)

Según el caso, el análisis de las imágenes requiere diferentes técnicas. En análisis de escenas dinámicas, las escenas SCMO son las que han recibido mayor atención. El análisis de estas escenas tiene usualmente como objetivo detectar movimiento, reconocer los objetos en movimiento y computar las características de sus movimientos.

Correspondencia

El proceso de correspondencia trata de identificar el mismo objeto en dos o más cuadros o imágenes y determinar cambios en su ubicación. En otras palabras, la correspondencia entre cuadros permite determinar el movimiento de los objetos.

Se trata entonces de aparear un punto $p_i = (x_i, y_i)$ de una imagen, con un punto $p_j = (x_j, y_j)$ de la siguiente de una secuencia de imágenes ordenadas temporalmente. Se puede pensar en estos puntos como las coordenadas que dan la ubicación de los objetos ubicados en los distintos cuadros. La disparidad entre los puntos está dada por el vector de desplazamiento entre ambos:

$$d_{ij} = (x_i - x_j, y_i - y_j)$$

El resultado de la correspondencia entre dos cuadros consecutivos es un conjunto de pares conjugados.

Para resolver el problema de correspondencia, se debe tener en claro algunos puntos:

- ¿Con qué criterio se afirma que dos puntos se corresponden? ¿Cuáles son las características que se comparan?
- ¿Qué limitaciones, si las hay, se imponen a los vectores de desplazamiento?

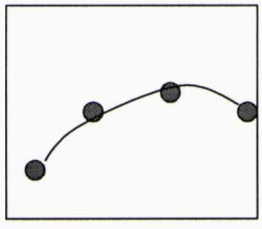
Como parámetros para responder a estas preguntas, es indispensable tener presente las siguientes propiedades fundamentales:

- *Similitud*: La similitud es una medida de cuánto dos puntos se asemejan entre sí. Como los puntos en este caso están representando objetos encontrados, la similitud va a estar

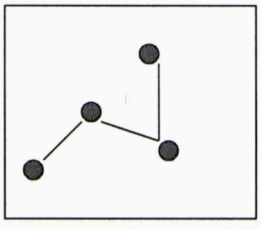
dada por la similitud entre los objetos. La misma se puede determinar a partir de las características de los objetos encontradas durante el proceso de segmentación.

- *Consistencia en el movimiento*: Debido a la inercia, el movimiento de una entidad física no puede cambiar instantáneamente. Por otro lado, se debe tener en cuenta que la proyección de una trayectoria tridimensional "suave" también lo es en el plano de una imagen de dos dimensiones. Por lo tanto si una secuencia de cuadros es obtenida con una frecuencia tal que entre dos cuadros consecutivos no tomen lugar cambios drásticos, entonces para la mayoría de los objetos físicos las imágenes van a reflejar la suavidad de los movimientos. Por lo tanto se puede asumir que el movimiento de un objeto en cualquier punto en una secuencia de cuadros no va a cambiar abruptamente.

En las siguientes figuras se puede observar esta segunda propiedad. Ambas representan el camino dibujado por el movimiento de un objeto. La trayectoria que muestra la figura (a) se asemeja bastante a la realidad. No ocurre lo mismo con la figura (b), donde el movimiento que se refleja no es creíble; se ve claramente la ausencia de consistencia en el recorrido.



(a)



(b)

Seguimiento de Trayectorias (Tracking)

Sea una secuencia de imágenes sobre las cuales se debe seguir una o más trayectorias. Si solamente hay un objeto en la secuencia de imágenes, el problema del seguimiento de su trayectoria puede resolverse fácilmente. En la presencia de muchas entidades moviéndose independientemente en una escena, el seguimiento de la trayectoria requiere el uso de restricciones basadas en la naturaleza de los objetos y sus movimientos. A partir de la propiedad de consistencia en el movimiento mencionada anteriormente, se puede formular la *coherencia del camino*. La coherencia del camino implica que el movimiento de un objeto en cualquier punto en una secuencia de cuadros no va a cambiar abruptamente.
Las siguientes tres suposiciones ayudan a formular una solución para resolver el problema de correspondencia:

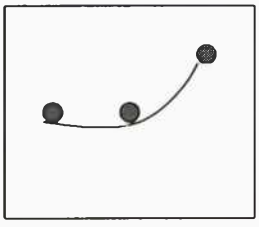
- La ubicación de un punto dado va a permanecer relativamente sin cambios de un cuadro al siguiente.
- La velocidad escalar de un punto dado va a permanecer relativamente sin cambios de un cuadro al siguiente.
- La dirección de movimiento de un punto dado va a permanecer relativamente sin cambios de un cuadro al siguiente.

Función de desviación o función de trayectoria

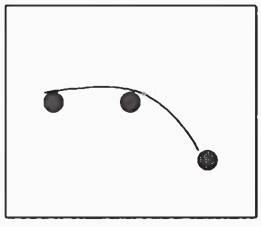
Para usar las propiedades descriptas arriba en un algoritmo, es necesario formular una función que implemente las ideas concretamente. Esta función a la que se llamará función de desviación o función de trayectoria, se usa para evaluar las propiedades del movimiento en una secuencia de cuadros. La idea es que reciba como entrada un trayecto y que el valor de

retorno sea directamente proporcional al grado de desviación del camino o, lo que es lo mismo, inversamente proporcional a la suavidad de la trayectoria. Sea el ejemplo ilustrado en el párrafo anterior. El objeto en movimiento de la imagen (b) toma un curso bastante deformado, por lo que para este caso la función debería retornar un valor mayor que para el trayecto ilustrado en la imagen (a), donde el trayecto es más armonioso.
Formalizar estas ideas, se tiene que la función de coherencia del camino debería cumplir con los siguientes principios:

1. El valor de la función debe ser siempre positivo. Cuanto más cercano a cero es el valor de retorno, mayor coherencia tiene el trayecto en cuestión.
2. Debería considerar la cantidad de desviación angular sin efecto del signo de la dirección del movimiento. Para el ejemplo ilustrado a continuación, la función debería retornar el mismo valor en ambos casos:



(a)



(b)

3. Debería responder de la misma manera a la velocidad incremental.
4. Debería ser normalizada en el rango 0.0 a 1.0.

Sea una trayectoria representada como

$$T_i = < P_i^1, P_i^2, P_i^3, ..., P_i^n >$$

donde T_i es la trayectoria y P_i^k representa un punto en la imagen *k*-ésima. Si las coordenadas del punto están dadas por el vector X_i del cuadro, las coordenadas del *k*-ésimo cuadro pueden ser representadas como X_{ik}. Representando la trayectoria en forma vectorial, tenemos:

$$T_i = < X_{i1}, X_{i2}, X_{i3}, ..., X_{in} >$$

Considérese ahora la desviación d_i^k en el camino, del punto en el *k*-ésimo cuadro. La desviación en el camino es una medida de la coherencia del camino, dada por

$$D_i^k = \phi(\overline{X_{ik-1}X_{ik}}, \overline{X_{ik}X_{ik+1}})$$

donde ϕ es una función de coherencia del camino. El valor resultante de la función da una idea de cuan coherente sería el movimiento de un objeto que pasa por estos puntos.
La desviación para toda la trayectoria se define como:

$$D_i = \sum_{k=2}^{n-1} d_i^k$$

Si hay *m* puntos en una secuencia de *n* cuadros, lo que resulta en *m* trayectorias, debería considerarse la desviación de todas las trayectorias, la cual está dada por:

$$D(T_1, T_2, T_3, ..., T_m) = \sum_{i=k}^{m} \sum_{k=2}^{n-1} d_i^k$$

De este modo, el problema de correspondencia se resuelve maximizando la suavidad del movimiento, o lo que es lo mismo, minimizando la desviación total *D* para encontrar el conjunto de trayectorias correctas.

Habiendo definido la función de trayectoria, se puede definir una función para la coherencia del camino. Si la frecuencia de muestreo de la cámara es suficientemente alta, entonces el cambio en la dirección y velocidad de cualquier punto en movimiento en cuadros temporalmente consecutivos es suave.

Esto se describe con la función de desviación:

$$\phi(P_i^{k-1}, P_i^k, P_i^{k+1}) = \omega_1(1 - \cos\theta) + \omega_2\left(1 - 2\frac{d_1 d_2}{d_1 + d_2}\right)$$

En forma vectorial puede ser representado como:

$$\phi(P_i^{k-1}, P_i^k, P_i^{k+1}) = \omega_1\left(1 - \frac{\overline{Xik\text{-}1Xik}\,\overline{XikXik+1}}{\|Xik\text{-}1Xik\|\,\|XikXik+1\|}\right) + \omega_2\left(1 - 2\frac{\sqrt{\|Xik\text{-}1Xik\|\,\|XikXik+1\|}}{\|Xik\text{-}1Xik\| + \|XikXik+1\|}\right)$$

El primer término es el producto vectorial de los vectores de desplazamiento y representa la *coherencia de dirección*. Esto es así ya que el producto vectorial es exactamente el ángulo entre los dos segmentos imaginarios entre los puntos k-1,k y k+1: Cuanto menor es el ángulo, mayor es la desviación.
El segundo término considera la media geométrica y aritmética de la magnitud, y representa la *coherencia de velocidad.*
ω_1 y ω_2 son pesos seleccionados para asignar diferente importancia a los cambios en dirección y velocidad. El peso puede seleccionarse en el rango de 0.00 a 1.00 tal que su suma de 1.

Una de las principales dificultades con el uso de múltiples cuadros es el problema de oclusión. Cuando se trabaja con una larga secuencia de cuadros, es posible que algunos objetos puedan desaparecer total o parcialmente. De la misma manera, pueden aparecer algunos objetos nuevos. Además, la forma cambiante de los objetos debido al movimiento y los cambios en la iluminación de la escena sobre la secuencia de cuadros, puede causar significantes cambios en los datos que tienen que corresponderse entre los cuadros. Todos estos cambios en el conjunto de características conducen a correspondencias incorrectas. Forzando las trayectorias para que satisfagan algunas restricciones locales y permitiendo que las mismas queden incompletas si es necesario, pueden obtenerse trayectorias en presencia de oclusión.

Coherencia del Camino en la presencia de oclusión

Dos limitaciones del algoritmo de coherencia del camino son que:

- Asume que el mismo número de puntos está disponible en todos los cuadros
- Asume que en todos los cuadros se extrae el mismo conjunto de puntos.

En la práctica, el número de objetos puede cambiar repentinamente de un cuadro al siguiente. Aún cuando la cantidad de objetos sea la misma sobre todos los cuadros, puede no

necesariamente ser el mismo conjunto que fue obtenido en los cuadros anteriores. De este modo, al buscar un conjunto suave de trayectorias, se debe permitir la posibilidad de obtener trayectorias incompletas que indiquen oclusión, aparición de un nuevo objeto, o simplemente la ausencia de los correspondientes objetos debido a una detección pobre de características. Más aún, para que una trayectoria sea aceptable, deben imponerse algunas restricciones sobre el máximo desplazamiento posible y la máxima desviación local; así se evitan las chances de agrupar puntos distantes como si se correspondieran.

Dado un conjunto P^j de m_j puntos para cada uno de los n cuadros, el siguiente algoritmo encuentra el máximo conjunto de trayectorias completas o parcialmente completas que minimiza la suma de desviaciones locales para todas las trayectorias obtenidas, sujeto a las condiciones de que la máxima desviación para cualquiera de las trayectorias no exceda ϕ_{max} y el desplazamiento entre cualquier par de cuadros sucesivos para cualquier trayectoria sea siempre menor que d_{max}. Estas restricciones limitan la ubicación aceptable de un punto en el siguiente cuadro dada su ubicación en los dos cuadros previos.

Algoritmo

Para justificar los puntos faltantes debido a la oclusión, se usan *puntos fantasma*. Los puntos fantasma son puntos hipotéticos que se usan como relleno para extender todas las trayectorias sobre el conjunto de cuadros dado. La noción de puntos fantasma sirve a dos propósitos: Permite satisfacer las restricciones locales y provee una manera para tratar con las trayectorias incompletas. Para que la noción de puntos fantasma sea útil, se deben definir los valores de desplazamiento y desviación local para una trayectoria que tiene puntos fantasma.

1) Par computar el desplazamiento para una trayectoria T_i en el k-ésimo cuadro se define la función de desplazamiento como sigue:

$$\text{Desp}\,(P_i^k, P_i^{k+1}) = \begin{cases} \text{Distancia Euclideana}\,(P_i^k, P_i^{k+1}) & \text{si ambos puntos son reales} \\ d_{max} & \text{en otro caso} \end{cases}$$

Esta definición implica que un punto fantasma siempre se mueve en una cantidad fija d_{max}.

2) Par computar la desviación local para una trayectoria T_i en el k-ésimo cuadro se define la función de desviación como sigue:

$$\text{Desv}\,(P_i^{k-1}, P_i^k, P_i^{k+1}) = \begin{cases} 0 & \text{si } P_i^{k-1} \text{ es un punto fantasma} \\ \phi\,(P_i^{k-1}, P_i^k, P_i^{k+1}) & \text{si los tres puntos son reales} \\ \phi_{max} & \text{en otro caso} \end{cases}$$

Esta definición de función de desviación local es equivalente a la función de coherencia del camino si los tres puntos son reales. Introduce una penalidad de ϕ_{max} en el caso de no tener un

punto real para T_i en el cuadro corriente o el siguiente, y no da penalidad si la trayectoria empieza en el cuadro que se está considerando.

Dadas las definiciones para computar el desplazamiento y la desviación local, se está en condiciones de plantear un algoritmo que realice el seguimiento de trayectorias. Básicamente se trata de un algoritmo que parte de un armado inicial tentativo de las trayectorias. Luego usa la función de desviación definida para intercambiar objetos entre esas trayectorias con el objetivo de lograr trayectorias con mínima desviación total.
El siguiente es un conjunto de pasos que muestran a grandes rasgos el procedimiento para el armado de las trayectorias.

Inicialización:

1. Para cada punto m_k en P^k, k=1, 2, ..., n-1 (P^k: conjunto de puntos del cuadro *k*), determinar el vecino más cercano en P^{k+1} que esté dentro de la distancia d_{max}. Resolver arbitrariamente en el caso de múltiples alternativas.
2. Formar trayectorias iniciales encadenando los vecinos cercanos encontrados en los cuadros sucesivos. Extender todas las trayectorias incompletas usando puntos fantasma para que vaya a lo largo de los *n* cuadros.
3. Para cada trayectoria del paso 2, formar una trayectoria adicional consistente sólo de puntos fantasma

Ciclo de intercambio:

Para cada cuadro desde k=2 hasta n-1
  Para i=1 hasta m-1
    Para j=i+1 hasta m
      Si se cumplen las restricciones de d_{max}, calcular

$$G_{ij}^{k} = [\phi(P_i^{k-1}, P_i^{k}, P_i^{k+1}) + \phi(P_j^{k-1}, P_j^{k}, P_j^{k+1})] - [\phi(P_i^{k-1}, P_i^{k}, P_j^{k+1}) + \phi(P_j^{k-1}, P_j^{k}, P_i^{k+1})]$$

  Tomar el par *ij* con máxima ganancia (G_{ij})
  Si la ganancia es mayor que cero entonces
    Intercambiar los puntos del cuadro *k+1*, P_i^{k+1} con P_j^{k+1}.

Terminación:

Repetir el ciclo de intercambio hasta que no haya más cuadros.

El número de trayectorias armadas durante el paso 2 de la inicialización depende de la calidad de los datos. En el caso ideal, donde el mismo número *m* de puntos está consistentemente presente en todos los cuadros, sólo se formarán *m* trayectorias y ninguna tendrá puntos fantasmas. En el peor de los casos, cuando los puntos de alguno de los cuadros no tengan ninguna correlación con los puntos de cualquier otro cuadro, el número de trayectorias que se forme puede ser tan grande como la suma total de todos los puntos de todos los cuadros. En general, el número de trayectorias va a ser al menos m_{max} donde $m_{max} = \max(m_1, m_2, \ldots, m_n)$ y las diferentes trayectorias van a tener diferentes números de puntos fantasma.
Nótese que la introducción de puntos fantasma no requiere las ubicaciones de estos puntos. La introducción de las trayectorias de puntos fantasma, asegura que ninguna trayectoria final va a tener una desviación local mayor que ϕ_{max} en ningún cuadro. El chequeo de los pares *ij* en el algoritmo asegura que la restricción de d_{max} no es violada.

Algunas consideraciones

Siempre se está suponiendo un caso donde hay varios objetos. En el caso particular en el que hay que seguir concretamente a un objeto conocido, una solución para economizar tiempo de procesamiento, es inspeccionar en cada imagen sólo la parte donde se estima que puede estar el objeto en función de su ubicación en los cuadros anteriores, y teniendo en cuenta el movimiento natural de los objetos.

Por otro lado en este algoritmo de seguimiento de trayectorias se está viendo a los objetos como un conjunto de cosas indivisibles. Una situación más interesante es aquella en la que se conoce el modelo del objeto o los objetos buscados. Entonces a medida que se inspecciona la imagen, se pueden ir descartando aquellos segmentos que no obedecen a ninguno de estos modelos.

Procesamiento Paralelo

Una de las motivaciones centrales del procesamiento paralelo es la replicación tanto de hardware como de software.
Considérese la replicación de software. Si el código para realizar una tarea ya existe, posiblemente una gran parte del mismo puede reusarse en una versión concurrente, es decir, para que el programa pueda ser ejecutado por los procesadores individuales; además no es necesario preparar un programa diferente para cada uno. Este estilo de programación como se verá más adelante es llamado SPMD (Single Program / Multiple Data). Cada procesador trabaja sobre un conjunto de datos local independiente, mientras que el programa usado es replicado – un único programa es ejecutado múltiples veces concurrentemente en diferentes conjuntos de datos.
Además de la reutilización de código, otra ventaja es que el costo de desarrollo de software no crece con el número de procesadores. Usualmente, de hecho, el número de procesadores a ser empleado es variable y se selecciona cada vez que se ejecuta el procedimiento según las dimensiones del problema en particular.
Un problema de las soluciones paralelas que no se presentan en una solución secuencial es el tema de la región fronteriza. El conjunto de datos o trabajo a ser cumplido puede particionarse, pero las piezas adyacentes deben ser compartidas por dos o más procesadores, y debe planificarse una manera de coordinar, intercambiar datos, y cooperar.
En una aplicación real, esto requiere que tanto los datos como el control o sincronización fluya a través de la frontera compartida por los procesadores. Esto agrega sobrecarga (overhead) tanto en tiempo para manejar datos, como en software adicional para controlar el intercambio de información. La cantidad de datos y la habilidad del hardware utilizado para mover los datos eficientemente van a ser factores importantes en el buen comportamiento o performance de una aplicación dada en procesamiento paralelo.
No se requiere una comunicación general, simplemente cada procesador necesita comunicarse sólo con sus vecinos. Esto es una ventaja significativa ya que el hardware sólo necesita proveer una capacidad de comunicación limitada y altamente estructurada más que una capacidad completamente genérica,
En general, los problemas que muestran patrones de comunicación localizados y estructurados, son perfectamente adecuados para una implementación con procesamiento paralelo. Cualquier máquina particular tiene una arquitectura subyacente donde algunos pares de procesadores están más cerca entre sí que otros. Si los requerimientos de comunicaciones lógicas del problema pueden ser mapeados a los procesadores físicos de manera tal que la distancia entre los procesadores que se comunican se minimiza, esto puede ser un aumento de performance significativo para una implementación paralela. Tal mapeo requiere no obstante el conocimiento por parte del programador del esquema de interconexión del hardware – hipercubo, malla, etc.
Afortunadamente la realidad es que muchos problemas exhiben requerimientos de comunicación extremadamente simples y altamente localizados. Esta característica puede tratarse de varias maneras. El procesamiento de imágenes por ejemplo naturalmente exhibe un arreglo bidimensional. Si se parte la imagen en cuadrados de igual superficie, cualquier subimagen va a tener cuatro vecinos (arriba, abajo, derecha e izquierda), u ocho si se requiere así. Una descomposición similar puede hacerse en las operaciones con matrices.
Una observación teórica de gran importancia al momento de tratar con procesamiento paralelo, es que la frontera o los bordes de un problema tienden a crecer significativamente menos rápido que su interior. El significado de esta observación es su impacto en la eficiencia. El problema a resolver tiene que ser suficientemente grande de modo que la

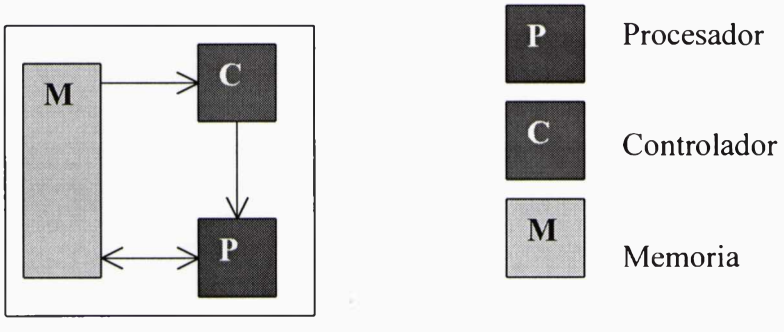
fracción de tiempo ocupada en overhead (la frontera de cada subproblema) sea pequeña con relación al problema en su totalidad.
Al tratar con procesamiento paralelo no deben descuidarse los subsistemas de entrada/salida. Típicamente a cada procesador se le suministra una vía de comunicación junto con la posibilidad de que muchos o todos los procesadores puedan manejar estos canales de comunicación simultáneamente. Otra solución a la entrada/salida es dedicar un subconjunto de los procesadores exclusivamente para las operaciones de entrada/salida al servicio del resto de los procesadores del conjunto.
Otras características claves para el éxito de una implementación de procesamiento paralelo incluyen:

- Broadcast, en el cual un controlador central se encarga de que un conjunto de datos esté disponible en todos los procesadores.
- Reducción, en la cual los datos o resultados de cada procesador se combinan (aplicando algún operador) y se entregan al controlador central.
- Sincronización, en la cual todos los procesadores deben alcanzar un punto en común en el procedimiento que están ejecutando, antes de proceder con el siguiente paso.
- Carga balanceada, en la cual se procura que el trabajo hecho por cada procesador sea semejante al trabajo hecho por cualquier otro, de modo que todos los procesadores se mantienen tan ocupados como sea posible durante todo el transcurso de la tarea que se está cumplimentando.
- Tarea no uniforme, donde a algún subconjunto pequeño de procesadores se le asigna una tarea sustancialmente diferente.

Arquitecturas De Hardware

Un multiprocesador consiste de un conjunto de procesadores que están distribuidos dentro de una única entidad, para realizar una o más funciones. El tipo de función necesaria determina la distribución seleccionada.

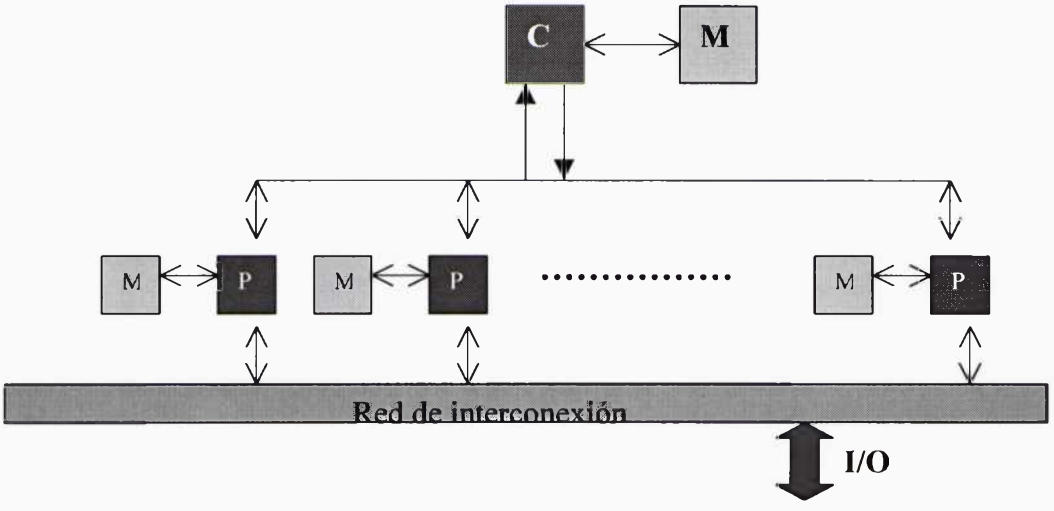
Para entender las distintas arquitecturas de multiprocesadores, es necesario partir de la estructura de una computadora secuencial, como la que se ilustra en la figura siguiente:



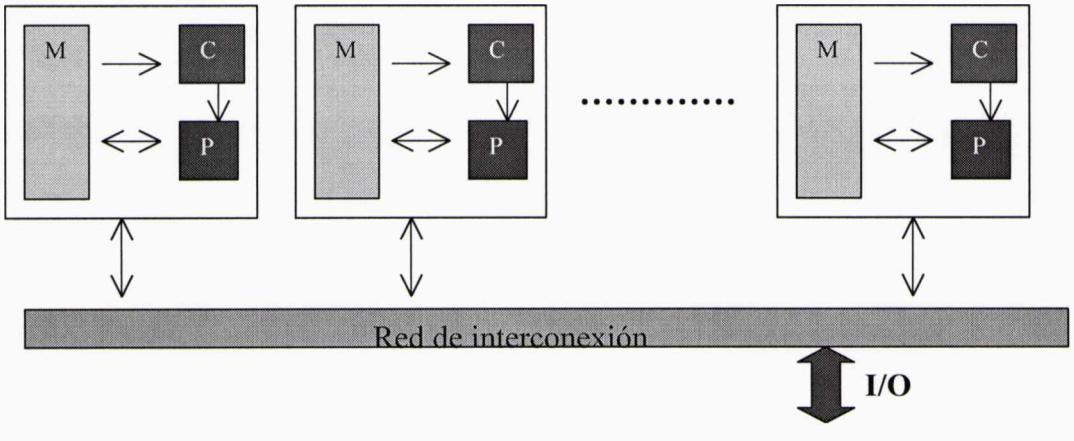
El almacenamiento o memoria (M) contiene las instrucciones para que el programa se ejecute, los datos sobre los cuales opera el programa y los datos producidos como salida del programa. El controlador (C) busca las instrucciones de la memoria y las decodifica. El resultado son señales enviadas al procesador (P). Este recibe las señales, mueve los datos desde y hacia la memoria y opera sobre los datos según las instrucciones dadas por la CPU.
Las arquitecturas de procesamiento paralelo se clasifican dependiendo del grado de paralelismo entre los conjuntos de datos e instrucciones, y de la organización de los

procesadores y la memoria. Todas pueden describirse disponiendo de varias maneras los tres componentes mencionados. Principalmente se tienen las tres clases siguientes:

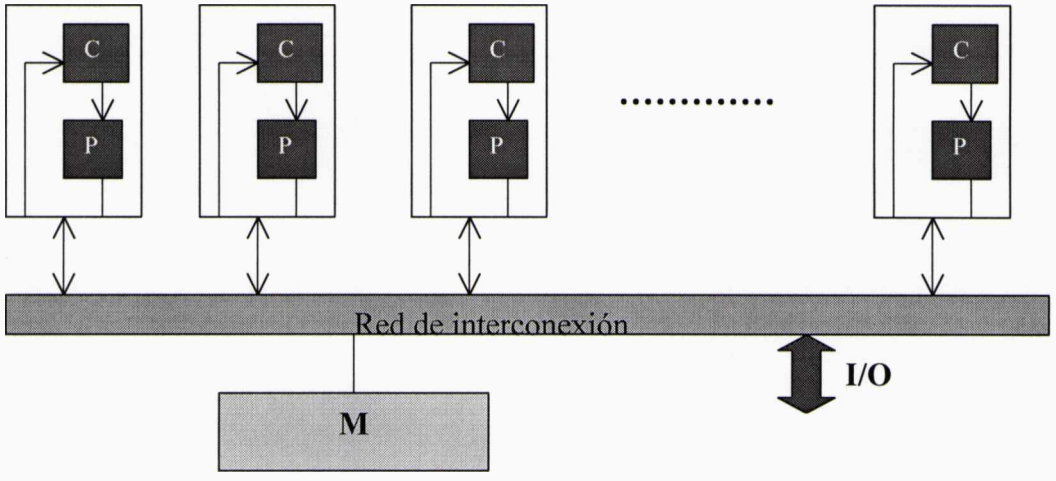
1) SIMD (Single Instruction Multiple Data) Idénticos procesadores con su propia memoria ejecutan las mismas instrucciones bajo el control de un único controlador maestro. El controlador transmite la instrucción a ser ejecutada a los procesadores paralelos simultáneamente. La siguiente figura muestra las relaciones de la instrucción y el conjunto de datos para esta arquitectura.



2) MIMD (Multiple Instruction Multiple Data) con memoria distribuida: No solo los procesadores y la memoria se replican, sino también los controladores, es decir que cada uno de los procesadores tiene su propio conjunto de instrucciones y de datos. Esta arquitectura es ilustrada en la figura siguiente:

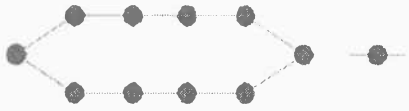


3) MIMD (Multiple Instruction Multiple Data) con memoria compartida: Se replican los procesadores y los controladores y se provee una única memoria accesible en forma uniforme. La memoria compartida es en particular usada como un mecanismo de comunicación entre procesos. Esta arquitectura se ilustra en la siguiente figura:

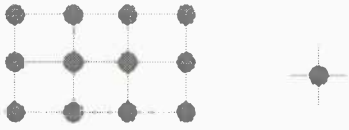


El usuario o programador de un multiprocesador tiene poco control sobre los aspectos de distribución pero puede hacer uso de ciertas técnicas de codificación para tomar ventaja de las características paralelas de la máquina. La tarea del desarrollador de una aplicación es seleccionar un algoritmo y una implementación de ese algoritmo que maximice el uso de las partes eficientes de la máquina y minimice el uso de las partes ineficientes. Por ejemplo, una máquina SIMD es naturalmente adecuada para las operaciones de broadcast y reducción. Estas mismas operaciones, no obstante, pueden tener una considerable sobrecarga cuando se ejecutan en una máquina MIMD de memoria distribuida.

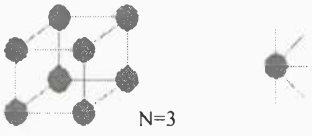
Independientemente de la clase de arquitectura que se presente, cada procesador paralelo tiene que estar conectado a una red de modo de poder compartir información con los otros procesadores. En muchas máquinas esta red es usada para soportar las operaciones de entrada/salida. Existen varias topologías o patrones usados para decidir qué pares de procesadores se conectan y cuáles no. Principalmente la topología usada es una de las siguientes: anillo, mallas de dos o tres dimensiones, o hipercubo. La figura siguiente muestra las distintas topologías junto con la conectividad, es decir cuántos vecinos tiene cada nodo dentro de la topología.



Anillo 2



Malla 4



Cubo N (2^N nodos)

Paradigmas De Programación Paralela

Comprendidas las principales arquitecturas de máquinas paralelas, las preguntas que surgen tienen que ver con los lenguajes utilizados para programar estas máquinas. Entre los diferentes paradigmas de programación existentes, los siguientes pueden considerarse como los más representativos de esta clase:

1. Memoria compartida
2. Pasaje de mensajes

Lenguajes de memoria compartida

La noción de variable local y variable compartida es fundamental para este paradigma. El hecho de que la memoria es uniformemente accesible por todos, es la base para la comunicación entre los distintos procesadores. Esta característica es implementada por medio de la posibilidad de designar ciertas variables como compartidas o *shared*. Por otra parte, no es deseable que todas las variables sean accedidas por todos los procesadores.

Todos los paradigmas de lenguaje de memoria compartida proveen mecanismos para distinguir entre variables locales y compartidas.

La coordinación de múltiples accesos a variables compartidas y la interacción entre variables locales y compartidas complican el ambiente. La manera más común de asegurar un acceso seguro a una variable compartidas es por medio de un lock. Todos los lenguajes paralelos para máquinas de memoria compartida proveen un mecanismo de lock.

Lenguajes de pasaje de mensajes

El pasaje de mensajes es la solución de programación natural para máquinas de memoria distribuida. Conceptualmente, el pasaje de mensajes consiste en tener cierto número de procesadores independientes, cada uno con su programa y sus datos privados. Los datos son

intercambiados entre estos procesadores por medio de mensajes enviados entre pares. Un mensaje es enviado por un nodo (*sender*), transmitido a través de la red, y eventualmente aceptado por el nodo destinatario (*receiver*). Para implementar este mecanismo de programación, el lenguaje debe proveer al menos las siguientes primitivas:

- send: para transmitir mensajes
- receive: para remover mensajes de la cola de los mensajes recibidos
- context: para permitir a un nodo conocer su número de identificación y el tamaño del arreglo de procesadores al que pertenece.

Usando estas primitivas, pueden construirse operaciones adicionales más elaboradas como por ejemplo *broadcast*, donde un nodo transmite el mismo mensaje a todos los demás.

Un estilo de programación frecuentemente usado para sistemas de pasaje de mensajes es SPMD. El mismo programa es ejecutado en cada uno de los nodos, y puesto que cada uno conoce su contexto, puede usar esta información para ramificar su curso dentro de la ejecución del código o para particionar grandes arreglos de datos.

Otro estilo también usado con frecuencia consiste en designar un único nodo para un papel especial. Este nodo actuaría como un maestro (*master*) repartiendo trabajo adecuadamente a los otros nodos, que harían el papel de esclavos (*slaves*). Este estilo requiere al menos dos módulos de código principales: El que ejecuta el maestro, y el que ejecutan los esclavos (este último típicamente replicado uno por procesador).

Message Passing Interfase (MPI)

El pasaje de mensajes como se vio en los párrafos anteriores, es un paradigma de programación ampliamente usado en computadoras paralelas especialmente en computadoras con memoria distribuida y en redes de estaciones de trabajo. Si bien hay muchas variantes, las bases de este paradigma son siempre las mismas. A partir de que varios sistemas han demostrado que el mecanismo de pasaje de mensajes puede ser implementado de manera portable y eficiente, nace MPI o Message Passing Interfase. MPI es un sistema de pasaje de mensajes estandarizado y portable diseñado por un grupo investigadores universitarios y de la industria. Los diseñadores de MPI intentaron hacer uso de las características más atractivas de un grupo de sistemas de pasaje de mensajes existente. Por otro lado se identificaron algunas deficiencias en los sistemas existentes, por ejemplo en el pasaje de mensajes con una estructura compleja o en la modularidad, lo cual llevó a la introducción de nuevas características propias de MPI.
El estándar MPI está pensado para ser usado por todo aquél que quiera escribir programas de pasaje de mensajes portables en "C" o en Fortran. Esto incluye tanto a programadores de aplicaciones individuales, como a desarrolladores de ambientes y herramientas.

Objetivos de MPI

El principal objetivo de MPI es la portabilidad a través de diferentes máquinas comparable con el grado de portabilidad de lenguajes como "C". Aunque normalmente se habla de pasaje de mensajes en el contexto de computadoras paralelas de memoria distribuida, el mismo código MPI puede ejecutarse sin inconvenientes en una computadora paralela de memoria compartida. Puede correr en una red de estaciones de trabajo o como un conjunto de procesos en una única estación de trabajo.
Otra característica de MPI es que ofrece la posibilidad de correr en forma transparente en sistemas heterogéneos, esto es, conjuntos de procesadores con distinta arquitectura. El usuario no necesita preocuparse si el código está enviando un mensaje entre procesadores de igual o diferente arquitectura, ya que la implementación hará automáticamente cualquier conversión de datos necesaria y utilizará el protocolo de comunicaciones correcto.
MPI no especifica cómo debe implementarse una operación; simplemente especifica la lógica de la operación. Como resultado, resta a cada implementación tomar ventaja de las características específicas de los subsistemas de comunicación de las distintas máquinas.
La escalabilidad es otro objetivo importante. MPI soporta la escalabilidad a través de varias de sus características. Por ejemplo, una aplicación puede crear subgrupos de procesos que a su vez permiten comunicaciones colectivas con un alcance limitado a los procesadores involucrados.

Plataformas para la Implementación

La interfaz MPI es adecuada para ser usada por programas MIMD (Multiple Instruction Multiple Data) o MPMD (Multiple Program Multiple Data), donde cada uno de los procesadores sigue un camino de ejecución distinto a través del mismo código o incluso puede ejecutar un código diferente. También es adecuado para ser escrito en un estilo más estricto como SPMD (Single Program Multiple Data) donde todos los procesos siguen el mismo camino de ejecución a través del mismo programa.
Existen varias implementaciones de MPI, entre ellas la implementación del Argonne National Laboratory y de Mississippi State University llamada MPICH la cual puede correr en muchos sistemas, y es la utilizada para la implementación llevada a cabo en el presente trabajo.

Características del Lenguaje

Conceptos Básicos

A través de MPI una aplicación ve su ambiente como un grupo de procesos. La colección inicial de procesos es llamada **world group**. A cada proceso miembro de este grupo se le asigna un único número llamado **rank**, que va desde 0 hasta N-1, donde N es el número total de procesos del **world group**. Los procesos pueden correr todos el mismo programa (SPMD) o diferentes programas (MIMD). Los procesos del **world group** pueden subdividirse creando subgrupos con un valor de **rank** potencialmente distinto en cada subgrupo.

El mecanismo básico de comunicación entre procesos es la transmisión de datos entre un par de procesos, uno que envía un mensaje y otro que lo recibe. Es lo que se llama comunicación *punto a punto*. Casi todas las construcciones del lenguaje están edificadas a partir de las operaciones punto a punto. Los datos que se envían son tipados y tienen asociados una etiqueta llamada **tag**. El **tag** es un entero arbitrario especificado por el programador. El **tag** permite la selectividad de mensajes en el receptor: Un proceso puede recibir mensajes con una determinada etiqueta, o puede especificar un comodín para poder recibir mensajes con cualquier valor en su etiqueta. También se permite la selectividad de los mensajes según el proceso origen del mismo.

MPI es tan simple que con sólo seis funciones se puede escribir una gran variedad de programas:

1. *MPI_Init*. Debe ser la primer función invocada por cualquier programa.
2. *MPI_Finalize*. Es la función por medio de la cual un proceso termina de realizar operaciones de MPI.
3. *MPI_Comm_Size*. Obtiene la cantidad de procesos de un grupo.
4. *MPI_Comm_rank* Obtiene el valor de **rank** del proceso que la invoca dentro de un grupo de procesos.
5. *MPI_Send*. Es la función por medio de la cual un proceso envía un mensaje a otro.
6. *MPI_Recv*. Es la función recíproca de la anterior; con esta función un proceso recibe un mensaje enviado por otro.

Ejemplos

A continuación se muestran dos ejemplos sencillos que ilustran la estructura de un programa MPI, y cómo se utilizan estas funciones elementales. El primero es el típico ejemplo "Hello, World". Debe notarse que todas las funciones no-MPI son locales, por lo cual *printf* se ejecuta en cada proceso. El segundo muestra un programa para correr exactamente con dos procesos. Un proceso envía un mensaje al otro y ambos terminan. (Para entender la sintaxis exacta de las operaciones, se puede consultar el apéndice respectivo)

```
/* Ejemplo 'Hello, World' */


#incude 'mpi,h'
#include <stdio.h>


int main(argc, argv)
int argc;
char **argv;
```

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

```
{
int rank,size;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank); /* consulta el rank propio */
MPI_Comm_size(MPI_COMM_WORLD, &size);  /* consulta el número de procesos */
printf('Hello, world! I'm %d of %d\n', rank, size);
MPI_Finalize();
Return 0;
}

/ * Transmisión de mensajes entre dos procesos */

#include 'mpi.h'
#define BUFSIZE 64
int buf[64];

int main(argc, argv)
int argc;
char **argv;
{
int rank,size;
MPI_Status status;

/*  Inicializa MPI  */
MPI_Init(&argc, &argv);

/*  Chequea el número de procesos. Determina el rank en el world group  */
MPI_Comm_size(MPI_COMM_WORLD, &size);
If (size != 2) {
        MPI_Finalize();
        return(1);
        }
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

/*  El que envía el mensaje va a ser el proceso 0 y el receptor el proceso 1  */
```

```
if (rank==0) {
        /* Envía el mensaje. Utiliza un valor arbitrario de tag; en este caso 11 */
        MPI_Send(buf, sizeof(buf), MPI_INT, 1, 11, MPI_COMM_WORLD);
        }
else    {
        MI_Recv(buf, sizeof(buf), MPI_INT, 0, 11, MPI_COMM_WORLD), &status);
        }
MPI_Finalize();
Return(0);
}
```

Comunicadores

Un comunicador es una estructura de datos de MPI que contiene información sobre un grupo. Los comunicadores son un argumento obligatorio de todas las rutinas de comunicación de MPI. Después de que un proceso es creado e inicializa MPI, tiene disponibles dos comunicadores:

1. MPI_COMM_WORLD que contiene a todos los procesos
2. MPI_COMM_SELF que representa el grupo unitario que como único miembro tiene al proceso en cuestión.

Muchas aplicaciones requieren únicamente el comunicador correspondiente al **world group**. Si se necesitan nuevos subgrupos, deben crearse nuevos comunicadores.

Comunicación bloqueante

El término *bloqueante* en MPI significa que la rutina no retorna hasta que el buffer de datos asociado pueda ser reusado.

El control de flujo y el uso de buffers presenta diferentes elecciones al diseñar las primitivas de pasaje de mensajes. MPI ofrece cuatro modos de transmisión que cubre las necesidades de sincronización, transferencia de datos y performance de la mayoría de las aplicaciones. El modo es seleccionado por el proceso que envía el mensaje por medio de cuatro funciones *send* distintas. Estos modos son:

1) **Standard**. La operación *send* se completa cuando el sistema puede colocar en un buffer el mensaje, o cuando el mensaje es recibido por el destinatario.
2) **Buffered**. La operación *send* termina cuando el mensaje es colocado en un espacio de buffer provisto por la aplicación o cuando el mensaje es recibido.
3) **Synchronous**. La operación *send* termina cuando el mensaje es recibido.
4) **Ready**. La operación *send* no debe empezar a menos que su correspondiente operación *receive* haya comenzado. Luego el *send* se completa en forma inmediata.

Comunicación no bloqueante

El término *no bloqueante* en MPI significa que el control del programa retorna inmediatamente y esto puede ocurrir aunque la operación de transmisión solamente haya empezado, no necesariamente terminado. Las distintas formas de la función *send* existentes para la comunicación bloqueante, tienen su contrapartida no bloqueante. Las operaciones no bloqueantes tienen un argumento extra: un objeto llamado **request**. Este objeto se utiliza posteriormente en una función utilizada para completar la operación. Recién cuando la operación es completada, el buffer de mensajes puede ser reutilizado.

El objetivo del pasaje de mensajes no bloqueante es empezar la transferencia de un mensaje lo antes posible, continuar con otras operaciones de computación importantes y luego tratar de completar la operación lo más tarde posible.

Tipos de Datos

MPI permite describir tipos de datos que van desde simples tipos primitivos hasta complejas estructuras, arreglos e índices.

Todas las rutinas de pasaje de mensajes aceptan un argumento *tipo de dato* que indica de qué tipo son los datos a enviar. Existen varios tipos de datos predefinidos los cuales cubren el conjunto de tipos de datos básicos disponibles en la mayoría de las arquitecturas, como son char, int o float.

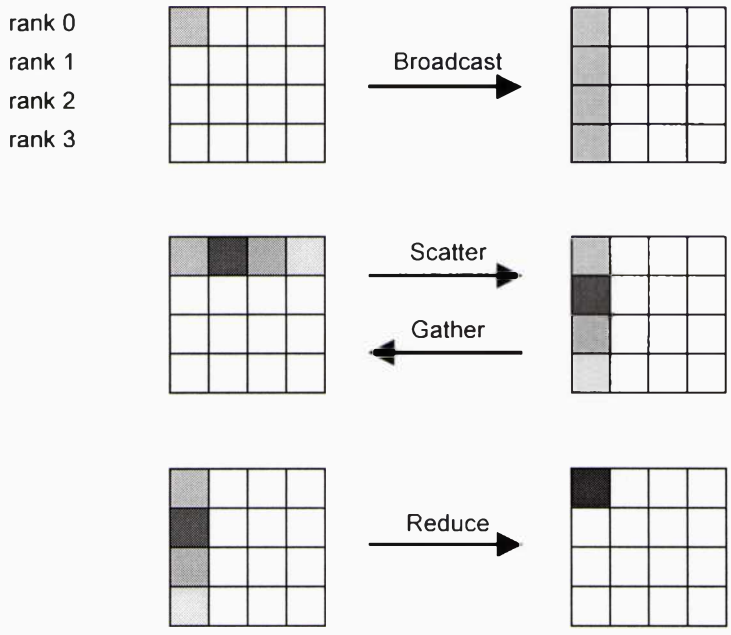
Con MPI se pueden construir tipos de datos derivados combinando tipos de datos básicos u otros tipos de datos derivados previamente construidos. A diferencia del tradicional concepto de tipo de dato definido por el usuario, en MPI un tipo de dato derivado describe un formato en memoria el cual consiste de múltiples arreglos de elementos con diferentes tamaños, distinto desplazamiento entre ellos, y cuyos elementos pueden ser a su vez de distintos tipos de datos.

Comunicaciones Colectivas

Las operaciones colectivas consisten de muchos mensajes punto a punto los cuales suceden concurrentemente e involucran a todos los procesos dentro de un comunicador dado. Cada proceso debe llamar a la misma rutina colectiva. La mayoría de las operaciones de este tipo son variantes y/o combinaciones de las siguientes cuatro primitivas:

1. *MPI_Bcast* (broadcast): En esta operación un proceso denominado raíz y que debe ser especificado por todos los procesos envía un mensaje a todos los procesos del grupo.
2. *MPI_Scatter*: Es otra operación uno a muchos, donde también se especifica un proceso raíz. Este proceso divide el buffer con el mensaje en partes iguales, que reparte entre todos los procesos - incluido sí mismo – en orden creciente de rank.
3. *MPI_Gather*: Es una operación colectiva muchos a uno y es exactamente la inversa de la operación anterior.
4. *MPI_Reduce*: La reducción consiste en realizar una operación de reducción global (tal como suma, máximo, AND lógico, etc.) entre todos los miembros de un grupo. La operación puede ser una de una lista predefinida o una operación definida por el usuario. Las operaciones definidas por el usuario deben cumplir la condición de ser asociativas, no necesariamente conmutativas.

Las distintas primitivas de comunicación colectiva son ilustradas en la siguiente figura:



El siguiente fragmento de código es un ejemplo sencillo que ilustra el uso de las operaciones colectivas primitivas. La cantidad de información a procesar es obtenida inicialmente por el proceso raíz y enviada a todos los procesos. El conjunto de datos inicial también es obtenido por el proceso raíz, el cual luego lo reparte entre todos los procesos. Después de la fase de computación, se computa un máximo global que es retornado al proceso raíz, y finalmente este proceso recopila el conjunto de datos modificados.

```
/* Ejemplo de utilización de operaciones colectivas */

... declaración de variables ...
root=0;
MPI_Comm_size(MPI_COMM_WORLD, &size);
MP_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank==root) {
        ... obtiene los datos a procesar ...
        }
MPI_Bcast(&longitud,1,MPI_INT,root,MPI_COMM_WORLD);
milongitud = longitud / size;
MPI_Scatter(datos, longitud, MPI_DOUBLE, misdatos, milongitud, MPI_DOUBLE, root,
MPI_COMM_WORLD);
... computación sobre los datos o obtención del máximo local 'maxlocal'...
MPI_Reduce(&maxlocal, &maxglobal, 1, MPI_DOUBLE, MPI_MAX, root, MPI_COMM_WORLD);
MPI_Gather(misdatos, milongitud, MPI_DOUBLE, datos, longitud, MPI_DOUBLE, root,
```

```
MPI_COMM_WORLD);
```

Timers y Sincronización

MPI define un cronómetro o timer. El timer pese a no ser estrictamente una herramienta relacionada con el pasaje de mensajes, es provisto dada la importancia de la medición de tiempos en programas paralelos a la hora de depurar y mejorar el rendimiento de los mismos. Se proveen dos funciones relacionadas con la medición del tiempo:

1. *MPI_Wtime*: retorna un número correspondiente a la cantidad de segundos transcurridos desde algún momento en el pasado.
2. *MPI_Wtick*: Retorna la resolución de *MPI_Wtime* en segundos. Es decir, retorna el número de segundos entre sucesivos ticks de reloj.

El siguiente ejemplo muestra una manera de usar el cronómetro.

```
{
double starttime, endtime;
starttime = MPI_Wtime();
. . . tareas cuyo tiempo se va a medir . . .
endtime = MPI_Wtime();
printf('La tarea tomó %f segundos\n', endtime · starttime);
}
```

Solución Del Sistema – Primera Aproximación

Introducción

Planteados los conceptos preliminares, se resolverá ahora el problema de reconocimiento de objetos y seguimiento de sus trayectorias por medio de una implementación concreta.

Estructura del Sistema y Fuente de los Datos
Pensando el sistema como un proyecto aplicable a la realidad, hay que pensar en una fuente de datos proveniente directamente de un dispositivo que capture las imágenes en tiempo real. A los fines de desarrollar y evaluar la implementación que acá se propone, la solución es diferente. Estos datos están disponibles completamente con anterioridad a la ejecución de la aplicación.

El siguiente algoritmo da una primera idea de lo que sería el ciclo principal de ejecución del sistema:

```
Mientras (hay cuadros)
        {
        Tomar un cuadro
        Identificar los objetos del cuadro
        }
Identificar y armar las trayectorias de los objetos hallados
```

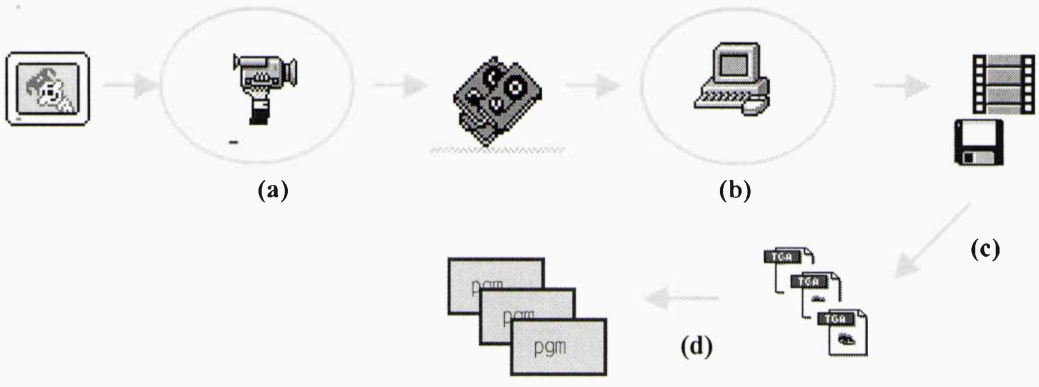
¿Qué significa "mientras hay cuadros"? En el caso que las imágenes provengan directamente de un dispositivo de obtención de datos en tiempo real, esto es equivalente a decir "mientras se estén captando imágenes". En el caso de imágenes disponibles con anterioridad, significa "mientras queden imágenes aún no procesadas". En esta implementación, para indicar dónde están las imágenes y determinar cuándo no hay más imágenes para procesar se proponen varias alternativas:

1. Todas las imágenes están en un mismo directorio. Para poder determinar la correcta secuencia temporal, estas imágenes tienen un nombre base seguido por un número que le da su posición en la secuencia, por ejemplo: foto01, foto02, etc.
2. La lista de imágenes está enumerada en un archivo y el orden temporal de las mismas está dado por el orden en que aparecen en ese archivo.
3. La lista de imágenes viene en la forma de una lista de parámetros de la línea de comandos.

Para generar un código donde la manipulación de imágenes sea sencilla y clara, se decidió trabajar con imágenes en formato **.pgm**. Estas imágenes están codificadas en la escala de grises de 0 a 255 y su formato es muy simple. Para trabajar con este tipo de imágenes se utilizaron funciones de librería provistas en la biblioteca PBMPLUS (para más detalle ver APENDICE B: REFERENCIA DEL FORMATO DE IMAGEN PGM (Portable graymap file format).
Las imágenes con las que se trabajó fueron obtenidas de la siguiente manera. Para cada secuencia de imágenes:

a) Se capturó la secuencia con una filmadora hogareña, obteniendo así un video analógico.

b) Se digitalizó el video por medio de un programa de software apropiado para ello, obteniendo como resultado un video digital (archivo en formato **.avi**)
c) Se convirtió este video digital en una secuencia de imágenes digitales con formato **.tga**.
d) Se convirtió cada una de estas imágenes con formato **.tga** en una imagen en formato **.pgm** por medio de una función específica para este fin, provista en la librería PBMPLUS.



Antes de comenzar a ver en detalle el desarrollo del proyecto, es importante hacer una aclaración. Considerando el volumen de datos que representa tener que manipular dos o tres imágenes simultáneamente, se decidió trabajar con las imágenes en archivo más que en memoria RAM. En memoria se mantiene solamente un subconjunto de los datos de las imágenes, que se renueva a medida que las operaciones lo van requiriendo. Hay que tener en cuenta que son varias las estructuras de datos que se necesitan manipular al mismo tiempo y que requieren el uso de memoria entre ellas las que contienen a las imágenes principales, a las etiquetas en el proceso de segmentación, a los objetos y a las trayectorias.

A continuación se explicará en detalle la implementación de las dos tareas principales del sistema:

- Reconocimiento de objetos
- Armado de las trayectorias de los objetos hallados

Por último se verá la manera en que se integran estas dos tareas, completando la explicación del funcionamiento de este sistema.

Parte I – Reconocimiento de Objetos

Realce de las Imágenes

Para lograr mejores resultados en el proceso de reconocimiento de objetos, lo ideal es mejorar previamente la calidad de las imágenes. Dado que las imágenes de muestra fueron capturadas específicamente para este fin, no existe el problema del deterioro o la mala calidad de las mismas. No obstante se notaron dos inconvenientes:
Primeramente, se vio que las imágenes obtenidas no mostraban una clara diferencia entre los objetos y el fondo tal como se esperaba. Esta falencia tuvo que ver con la pobre iluminación en el momento de la captura.

Por otro lado se observó que ante un movimiento acelerado de los objetos, los cuadros a lo largo de la secuencia se veían inevitablemente borrosos.
Estos dos puntos originaron que este paso de realce de la imagen tenga como fin dar mayor definición a la imagen, al tiempo que aumente el contraste acentuando las formas y facilitando así el posterior reconocimiento.
Concretamente lo que se hizo fue primero calcular un promedio espacial con el fin de dar forma a los objetos con poca definición. Para ello se utilizó la ventana de promedio espacial de cinco puntos del ejemplo dado en la teoría:

0	1/8	0
1/8	1/4	1/8
0	1/8	0

Para completar la operación, se destacaron los puntos pertenecientes a objetos por medio de la siguiente función de ensanchamiento del contraste:

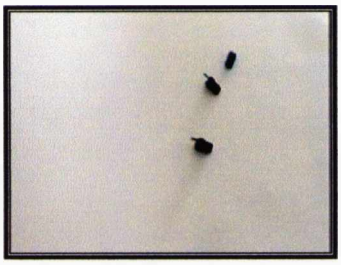
$$\begin{cases} -u/2 & si\ 0 < u < 128 \\ -(u-128)/2 + 192 & si\ u \ (128 \end{cases}$$

Siendo A la imagen original y B la imagen resultante, y *n* y *m* las dimensiones de la imagen, y *contraste* la función explicada, el ciclo principal de esta operación es el siguiente:

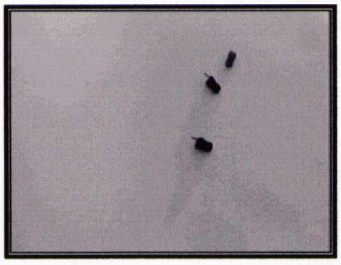
```
Obtener la primer fila de la imagen
Para j=1 hasta m          // La primer fila queda igual
        B(1, j) = contraste(A(1, j))

Obtener la segunda fila de la imagen

Para i=3 hasta n {

        Obtener la fila i de la imagen

        k = i – 1

        B(k,1) = contraste(A (k,1))

        Para j=2 hasta m-1
                B(k, j)=contraste(1/8 A(k, j-1) + 1/8 A(k-1, j) + 1/4 A(k, j) + 1/8 A(k+1,j) + 1/8 A(k, j+1))

        B(k, m) = contraste(A (k, m))

        }

Para j=1 hasta m          // La última fila queda igual
        B(n, j) = contraste(A(n, j))
```

Así, el valor de cada punto (fila, columna) de la imagen fue reemplazado por el resultado de la función de contraste aplicada al promedio pesado dado por la matriz.

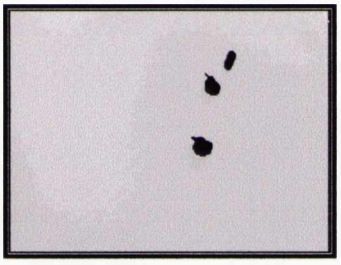
Notar que a la primera y última fila no se les aplica el promedio. Esto es porque para estas filas no se cuenta con puntos vecinos en todas las direcciones par poder realizar los cálculos. Lo mismo ocurre con el primer y último punto de cada fila.



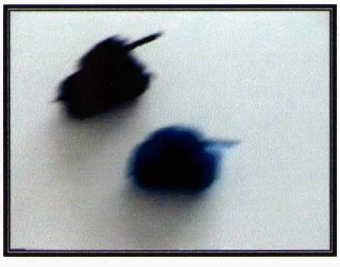
a) Imagen original



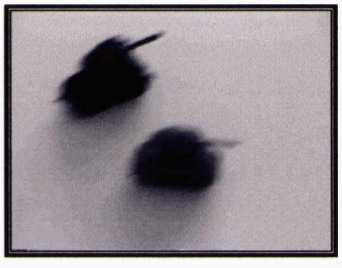
b) Imagen en formato pgm



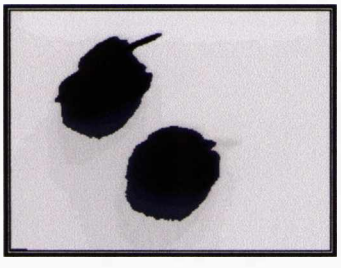
c) Imagen realzada



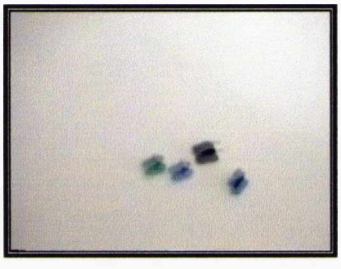
a) Imagen original



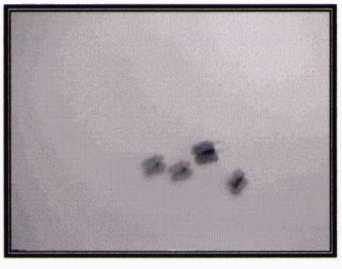
b) Imagen en formato pgm



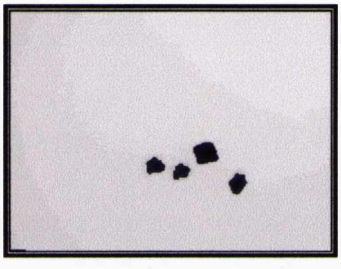
c) Imagen realzada



a) Imagen original



b) Imagen en formato pgm



c) Imagen realzada

Ejemplos de Realce de Imágenes

Reconocimiento de Objetos

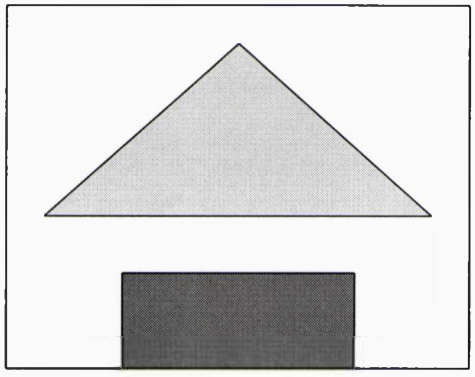
Para hacer esta parte del proceso se realiza un recorrido de la imagen de izquierda a derecha y de arriba hacia abajo - el sentido en el que usualmente están grabadas las imágenes en un archivo y en particular las imágenes **.pgm** -, analizando la información de sus pixels. Al terminar el recorrido de la imagen, y luego de haber depurado la información como se explica más adelante, se obtiene como resultado una lista de los objetos encontrados en la escena con las características que los identifican. Un primer refinamiento del algoritmo de reconocimiento sería:

```
Para i=1 hasta n (número de filas)
        Para j=1 hasta m (número de columnas)
                {
                tomar el punto Pij
                ver si corresponde al área de los objetos
                si es así analizar a que objeto pertenece y actualizar los datos de ese objeto
                }
Descartar información no relevante
```

Dado a que en esta implementación el foco se centrará en el seguimiento de trayectorias más que en el reconocimiento de objetos, se supondrá que los objetos son bastante distinguibles del fondo (background) de las imágenes, y que las características de interés son mínimas:

- ubicación
- área
- intensidad promedio (nivel de gris o color)

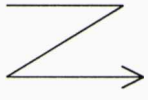
Además estos atributos son suficientes para caracterizar a los objetos, dado que no se buscan objetos con determinadas características; la idea en cambio es encontrar cualquier objeto que se encuentre en la escena.
La técnica utilizada para la identificación de los objetos dentro de una imagen combina el **thresholdind** con el **labeling**, técnicas ya descriptas (ver "Segmentación" en la página 7).
Primeramente se establece un threshold (ver "Selección del Threshold" Más adelante). Basándose en ese threshold se clasifican los distintos puntos de la imagen como pertenecientes o no a un objeto. Se aprovecha en este caso el conocimiento previo, que es como se mencionó, el hecho de que el fondo de las imágenes es parejo y contrasta con la parte ocupada por los objetos. Para comprender la técnica sea la siguiente imagen de ejemplo, junto con su representación en tonos de gris:



0	0	0	0	0	0	0	0	0
0	0	0	0	**105**	0	0	0	0
0	0	0	**105**	**105**	**105**	0	0	0
0	0	**105**	**105**	**105**	**105**	**105**	0	0
0	**105**	**105**	**105**	**105**	**105**	**105**	**105**	0
0	0	0	0	0	0	0	0	0
0	0	**209**	**209**	**209**	**209**	**209**	0	0
0	0	**209**	**209**	**209**	**209**	**209**	0	0

La técnica consiste en rastrear la imagen analizando cada punto o pixel, y comparando con los puntos vecinos. Por "punto vecino" se entiende todo punto que está a una distancia predeterminada de filas o columnas del punto en cuestión. En este caso se consideró vecino a cada punto distante una celda del punto actual. Cada vez que se detecta una diferencia significativa de intensidad entre el punto actual y un punto vecino, se guarda el dato, ya que esto es una indicación de que en este sector de la imagen puede haber un objeto.
Aplicar este tratamiento a la imagen, genera como resultado un dibujo o representación binaria de la imagen, donde una parte representa al fondo, y la otra corresponde a los diferentes objetos. Para el ejemplo, si se representase la zona de objetos con un "1" y el fondo con un "0", el resultado sería el siguiente:

0	0	0	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0
0	0	0	**1**	**1**	**1**	0	0	0
0	0	**1**	**1**	**1**	**1**	**1**	0	0
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
0	0	0	0	0	0	0	0	0
0	0	**1**	**1**	**1**	**1**	**1**	0	0
0	0	**1**	**1**	**1**	**1**	**1**	0	0



Sentido de recorrido de la imagen

Hasta este momento, se identificó la zona ocupada por objetos. Para completar la técnica, se estudia la conectividad de los puntos pertenecientes a la región de los objetos. Esto permite que lo que en un principio es una única región, se vea dividida en los distintos objetos que componen la escena, cada uno identificado por una etiqueta o label diferente. El resultado final para el ejemplo es el siguiente:

0	0	0	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0
0	0	0	**1**	**1**	**1**	0	0	0
0	0	**1**	**1**	**1**	**1**	**1**	0	0
0	**1**	**1**	1	**1**	**1**	**1**	**1**	0
0	0	0	0	0	0	0	0	0
0	0	**2**	**2**	**2**	**2**	**2**	0	0
0	0	**2**	**2**	**2**	**2**	**2**	0	0

Una vez finalizado el recorrido de toda la imagen, resta depurar la información obtenida. Por ejemplo puede ser que se hayan encontrado segmentos tan pequeños o tan débiles en su color, que no merezcan que se los considere como objetos ya que simplemente pueden tratarse de una simple mancha, o una sombra. Por lo tanto un parámetro decisivo al momento de realizar la identificación de objetos es el valor utilizado como un tope mínimo para considerar a un segmento como un verdadero objeto de la escena.

En particular para los casos de prueba del presente proyecto se estableció un tope mínimo igual al 0,005% de la superficie de la imagen. Por ejemplo, para las imágenes de 320 x 240 pixels, se fijó como tope 320 * 240 * 0,00005 = 3,84. Es decir, cualquier mancha que el algoritmo pudiera distinguir y cuya superficie sea menor a 4 pixels, no será tenida en cuenta como objeto de la imagen estudiada.

Planteada la técnica utilizada para la identificación de los objetos, el algoritmo anterior queda modificado de la siguiente manera:

```
Calcular el valor del threshold a utilizar (t)
Para i=1 hasta n (número de filas)
        Para j=1 hasta m (número de columnas)
                {
                tomar el punto Pij
                comparar su valor de intensidad de gris contra t
                si es mayor
                        analizar la conectividad con sus vecinos para ver a qué objeto pertenece
                                si pertenece a un único objeto ya encontrado
                                        colocar la etiqueta correspondiente a ese objeto
                                        actualizar los datos del objeto con los datos del punto Pij
                                si no, si pertenece a varios objetos candidatos
                                        combinar la información de los objetos, dejando uno solo
                                        colocar la etiqueta correspondiente a ese objeto
                                        actualizar los datos del objeto con los datos del punto Pij
                                si no
                                        asociarle una etiqueta nueva
                                        crear un objeto nuevo con los datos del punto Pij
                }
Para cada objeto encontrado
        Si no cumple con  los requisitos predeterminados para ser un objeto de interés
                Descartar el objeto
```

Como se ve en el algoritmo, los pasos de thresholding y labeling no se realizan por separado; un solo recorrido de la imagen alcanza para obtener el resultado buscado.
Se debe notar que una restricción impuesta a las imágenes es que en ella los objetos no se toquen. De otra manera, no sería éste el algoritmo apropiado.

Estructuras de Datos

La información de objetos y etiquetas se mantiene mediante dos archivos: El primero contiene la información de las etiquetas o "labels" que indican los lugares donde la imagen muestra una intensidad destacada. El otro archivo tiene la información de los objetos que se van encontrando; aquí se van recopilando los datos relacionados con las características de los objetos por ejemplo, la intensidad del nivel de gris. Cada segmento u objeto encontrado en la imagen, se refleja por lo tanto en ambos archivos. Al terminar el procedimiento se tiene en los dos archivos, el de etiquetas y el de objetos, la información esperada.

A continuación se especifican en detalle las estructuras de datos correspondientes.

- Archivo de Etiquetas: Contiene un registro por cada pixel de la imagen. Está ordenado de la misma manera que las imágenes originales, es decir por filas y dentro de cada fila por columnas. Para cada pixel contiene un 0 si ese elemento pertenece al fondo de la imagen, o el valor de etiqueta correspondiente, si el elemento pertenece a alguno de los objetos.
- Archivo de Objetos: Contiene la información de los objetos encontrados. Para cada objeto, se guarda un registro con la siguiente información:
  - posición (fila y columna dentro de la imagen correspondiente al centro del objeto)
  - área que ocupa
  - intensidad promedio

Hay tantas etiquetas distintas en el primer archivo como objetos hay en el segundo. La relación de una etiqueta con un objeto está dada por la posición de este dentro de su archivo, por ejemplo, los datos del objeto etiquetado con el número 2 están descriptos por el segundo objeto del archivo de objetos.

Para completar el ejemplo dado, se transcribe acá lo que sería el contenido de los archivos de referencia:
Archivo de etiquetas:
0,0,0,0,0,0,0,0,0,0, 0,0,0,0,1,0,0,0,0,0, 0,0,0,1,1,1,0,0,0, 0,0,1,1,1,1,1,0,0, 0,1,1,1,1,1,1,1,0,
0,0,0,0,0,0,0,0,0,0,
0,0,2,2,2,2,2,0,0, 0,0,2,2,2,2,2,0,0

Archivo de objetos:
[(4,5),16,105], [(8,5),10,209]

Selección del Threshold
Como se vio anteriormente, existen diversas técnicas para la búsqueda de este valor denominado threshold. En esta implementación se decidió utilizar un método de selección iterativo. Este método empieza con un threshold aproximado y luego esta estimación se va refinando sucesivamente. Se espera que algunas propiedades de las subimágenes resultantes del procedimiento de thresholding puedan ser usadas para seleccionar un nuevo valor de threshold que en cada iteración particione la imagen mejor que en el ciclo anterior. La

iteración continua hasta que el valor obtenido se estabiliza. Los siguientes pasos muestran el método:

1. Seleccionar un valor estimado inicial de threshold, T.
2. Particionar la imagen en dos grupos, R_1 y R_2, usando el threshold T.
3. Calcular la media de los valores de gris μ_1 y μ_2 de las particiones R_1 y R_2.
4. Seleccionar un nuevo threshold:
   $T = (\mu_1 + \mu_2) / 2$
5. Repetir los pasos 2-4 hasta que los valores de μ_1 y μ_2 en sucesivas iteraciones permanezcan sin cambios.

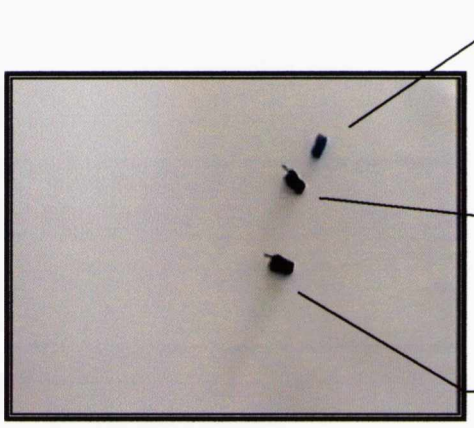
En este caso se tomó como valor inicial el que el método sugiere: La intensidad promedio de la imagen. El algoritmo correspondiente resulta de la siguiente manera:

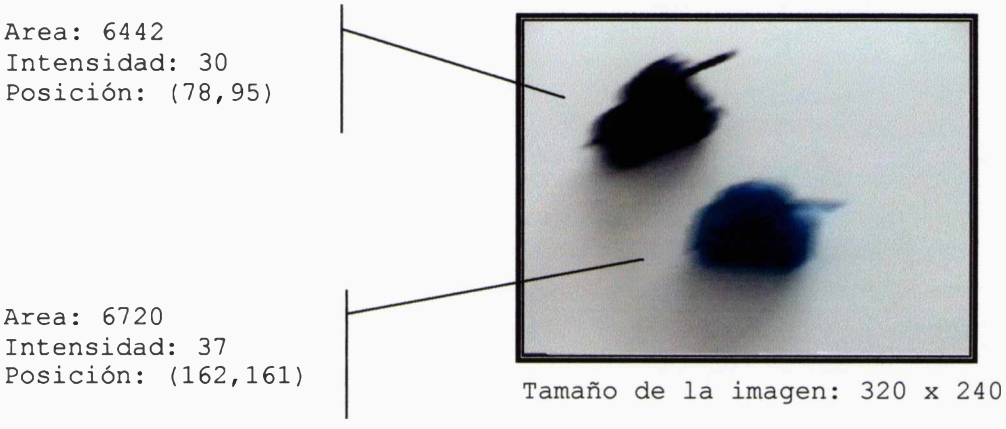
```
/*  Variables:
    m1Anterior, m2Anterior: los valores medios de las regiones en el ciclo anterior.
    threshold: el valor del threshold que se esta calculando
    R1, R2: la suma de los valores de los puntos en las dos regiones
    cantPuntos1, cantPuntos2: La cantidad de puntos en las regiones R1 y R2 respectivamente
    A(i, j): El valor del punto de la imagen en la posición (i, j)

// Calcula la media de la imagen
Total = 0
Para i = 1 hasta n
        Para j = 1 hasta m
                Total = Total + A (i, j)
laMedia = Total / ( n * m )

// Busca el valor de threshold
m1Anterior = m2Anterior = 0
threshold = m1 = m2 = laMedia
Mientras (m1Anterior <> m1 y m2Anterior <> m2)
        {
        R1 = R2 = cantPuntos1 =  cantPuntos2 = 0
        Para i=1 hasta n
                Para j=1 hasta m
                        Si A(i, j) < threshold entonces
                                R1 = R1 + A(i, j)
                                cantPuntos1 = cantPuntos1 + 1
                        Si no
                                R2 = R2 + A(i, j)
                                cantPuntos2 = cantPuntos2 + 1
        m1Anterior = m1
        m2Anterior = m2
        m1 = R1 / cantPuntos1
        m2 = R2 / cantPuntos2
        threshold = (m1 + m2) / 2
        }
```

Sugerencias

En este caso se conoce que en la escena hay varios objetos, y el objetivo final es encontrar la trayectoria de sus movimientos. Los objetos son vistos como un conjunto, y todos son parte

del proyecto; no son objetos indivisibles uno del otro. La situación por lo tanto es muy general. Para el caso de tener situaciones más específicas, se dan acá un par de consideraciones a tener en cuenta. En el caso particular en el que hay que seguir a un único objeto, una mejora para economizar tiempo de procesamiento es inspeccionar en cada imagen sólo la parte donde se estima que puede estar el objeto en función de su ubicación en los cuadros anteriores. Teniendo en cuenta que las imágenes provienen de la realidad, no tiene sentido buscar un objeto muy lejos de donde apareció en el cuadro precedente.
Una situación más interesante es aquella en la que se conoce el modelo del objeto o los objetos buscados. Entonces a medida que se inspecciona la imagen, se pueden ir descartando aquellos segmentos que por sus características se ve que no obedecen a ninguno de estos modelos.





Ejemplos de reconocimiento de objetos

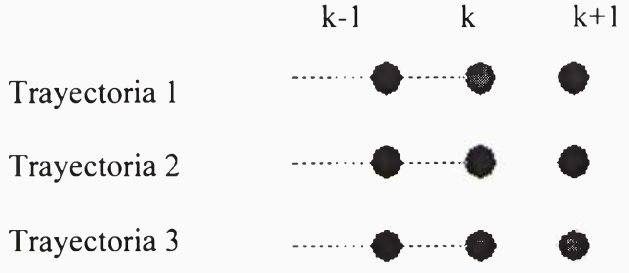
Parte II – Seguimiento de Trayectorias

Ahora se verá cómo tratar en la práctica el tema de la correspondencia de objetos y el seguimiento de sus trayectorias. En todo el desarrollo de esta implementación se supondrá que la secuencia de imágenes corresponde a una filmación donde la cámara es fija, y que el movimiento de los objetos es paralelo al plano de la imagen.

En teoría se vio un algoritmo genérico de cómo encarar el armado de las trayectorias. En ese planteo se suponía que toda la secuencia de imágenes estaba disponible desde un principio. Así, el proceso comenzaba con el armado de trayectorias tentativas que se extendían a lo largo de toda la secuencia de imágenes, y luego estas trayectorias eran modificadas siguiendo ciertas pautas hasta que se obtenían las trayectorias definitivas. En este caso, como se pretende definir un algoritmo útil para el caso de la obtención de las imágenes en tiempo real, tal suposición no es válida. Por lo tanto el algoritmo se ve necesariamente modificado para que funcione aunque las imágenes se vayan incorporando con el tiempo. La idea es que las trayectorias se vayan ampliando a medida que se va disponiendo de las imágenes. Básicamente el algoritmo principal se modifica como sigue:

```
Mientras (hay cuadros)
        {
        Tomar un cuadro
        Identificar los objetos del cuadro
        ACTUALIZAR LAS TRAYECTORIAS DE LOS OBJETOS HALLADOS (*)
        }
```

Es precisamente la línea de actualización de las trayectorias (*) lo que se analizará ahora.
Imagínese que el cuadro actual de la secuencia de imágenes es el número k y que en cada uno de los cuadros anteriores se han identificado P1, P2 . . .Pk-1 objetos. Hasta el momento se tiene armadas n trayectorias que van desde el cuadro 1 hasta el cuadro o imagen k-1. La idea es que el procedimiento asocie en la medida de lo posible, uno a uno los objetos del cuadro actual con cada una de las trayectorias que se están armando.
Sea el caso en que el número de objetos coincida con el número de trayectorias que se tienen hasta el momento. Se está frente al caso más simple y quizás el más probable. La situación se muestra gráficamente en el siguiente esquema:



Los círculos representan los objetos encontrados en los cuadros k-1, k y k+1, y las líneas punteadas, las trayectorias ya identificadas. Los objetos del nuevo cuadro, es decir, los

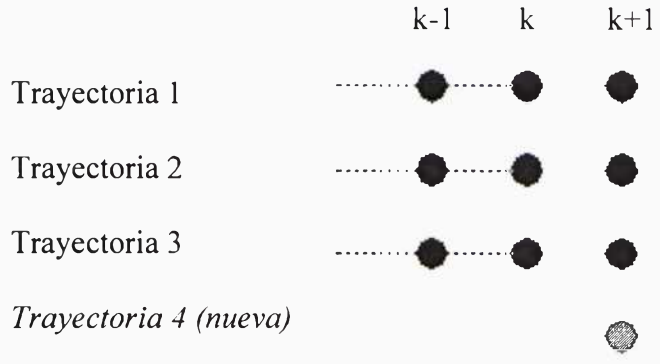
círculos de la última columna, son los próximos objetos a encadenar a las trayectorias ya existentes.

El algoritmo que resuelve el problema es en principio como sigue:

```
// Calcula la máxima desviación global
Para i=1 hasta m-1
        Para j=i+1 hasta m
                calcular G_ij^k = [φ(P_i^{k-1}, P_i^k, P_i^{k+1}) + φ (P_j^{k-1}, P_j^k, P_j^{k+1})] –
                                  [φ (P_i^{k-1}, P_i^k, P_j^{k+1}) + φ (P_j^{k-1}, P_j^k, P_i^{k+1})]
                Tomar el par ij con máxima ganancia (G_ij)
Si la ganancia G_ij es mayor que cero entonces
        Intercambiar los puntos del cuadro k+1, P_i^{k+1} con P_j^{k+1}.
Grabar el resultado de este paso en el archivo de trayectorias
```

En el ejemplo dado, la cantidad de objetos **m** en todos los cuadros es 3. Recordar que ϕ es la simbología para la función de desviación, y G_{ij} corresponde a la ganancia de intercambiar puntos en dos trayectorias.

Este algoritmo no presenta problemas excepto que en la secuencia aparezca un objeto o se deje de ver temporal o definitivamente otro. Sea el caso que en la nueva imagen haya más objetos que el número de trayectorias que se tienen hasta el momento. Entonces se debe interpretar que aparece un nuevo objeto en la escena y entonces crear una nueva trayectoria. El siguiente dibujo muestra la situación:

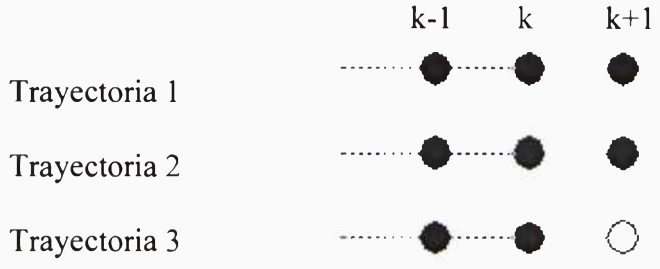


Para contemplar este caso, es necesario crear los llamados "puntos fantasma" o imaginarios en los cuadros k-1 y k. De esta manera se asegura que la función de desviación tenga todos los parámetros que necesita.
El algoritmo anterior queda modificado como sigue:

```
Extender todas las trayectorias incompletas usando puntos fantasma para que vaya a lo largo de los
3 cuadros recientes

// Calcula la máxima desviación global
Para i=1 hasta m-1
```

Para j=i+1 hasta m
calcular $G_{ij}^{k} = [\phi(P_i^{k-1}, P_i^{k}, P_i^{k+1}) + \phi(P_j^{k-1}, P_j^{k}, P_j^{k+1})] - [\phi(P_i^{k-1}, P_i^{k}, P_j^{k+1}) + \phi(P_j^{k-1}, P_j^{k}, P_i^{k+1})]$
Tomar el par *ij* con máxima ganancia (G_{ij})
Si la ganancia G_{ij} es mayor que cero entonces
Intercambiar los puntos del cuadro *k+1*: P_i^{k+1} con P_j^{k+1}
Grabar el resultado de este paso en el archivo de trayectorias

Ahora sea el caso opuesto. El número de objetos es menor que la cantidad de trayectorias que se tienen. En esas imágenes donde un objeto no aparece, se hace una estimación de su posición, en función de la que ocupaba en los cuadros precedentes. Nuevamente esto implica crear un punto fantasma para el nuevo cuadro. Observando la primer línea del algoritmo para el caso anterior, se ve que este caso también está contemplado.
En la figura siguiente el círculo blanco representa un punto fantasma.



Ahora, qué pasa cuando, a pesar que la cantidad de objetos del nuevo cuadro coincide con el número de trayectorias creadas, las trayectorias no respetan las pautas establecidas? Esto es, existe alguna terna de puntos en una o más trayectorias tal que la desviación del camino supera el mínimo predeterminado. En ese caso es necesario crear trayectorias nuevas formadas exclusivamente por puntos fantasma. Así, en el momento de evaluar la desviación y el intercambio de los puntos, una o más de las trayectorias existentes van a extenderse con uno de esos puntos fantasma y en el lugar que estos puntos ocupaban va a quedar destinado para esos objetos que violaban la restricción de la distancia máxima.

Con estas últimas consideraciones, el algoritmo finalmente queda como sigue:

Para cada objeto existente en el cuadro k+1 crear una trayectoria fantasma

// Calcula la máxima desviación global
Para i=1 hasta m-1
Para j=i+1 hasta m
Si en el intercambio se cumplen las restricciones de d_{max}, calcular
$G_{ij}^{k} = [\phi(P_i^{k-1}, P_i^{k}, P_i^{k+1}) + \phi(P_j^{k-1}, P_j^{k}, P_j^{k+1})] - [\phi(P_i^{k-1}, P_i^{k}, P_j^{k+1}) + \phi(P_j^{k-1}, P_j^{k}, P_i^{k+1})]$
Tomar el par *ij* con máxima ganancia (G_{ij})

Si la ganancia G_{ij} es mayor que cero entonces
    Intercambiar los puntos del cuadro *k+1*, P_i^{k+1} con P_j^{k+1}.
Grabar el resultado de este paso en el archivo de trayectorias

Para la solución dada - tal como se mencionó en la parte I - la correspondencia se basa exclusivamente en la ubicación de los objetos, es decir, se busca que se respete una consistencia en la dirección y velocidad de los objetos. Una solución más específica consistiría en buscar similitud en los objetos de cuadros consecutivos, usando las características encontradas en el proceso de reconocimiento de objetos.

Función de Desviación
Recuérdese la función de desviación ya vista:

$$\phi(P_i^{k-1}, P_i^k, P_i^{k+1}) = \omega_1(1 - \cos\theta) + \omega_2\left(1 - 2\frac{d_1 d_2}{d_1 + d_2}\right)$$

Como se deduce de esta definición, para implementarla es necesario tomar un par de decisiones:

1) Qué método utilizar para tomar la distancia entre dos puntos. En este caso se utilizó el método llamado *city-block*, donde la distancia entre los puntos (i_1,j_1) y (i_2,j_2) se define como:
   $d = |i_1 - i_2| + |j_1 - j_2|$
2) Pesos utilizados para la coherencia de dirección y la coherencia de velocidad en la fórmula, ω_1 y ω_2. En este caso no se dio preferencia a ninguna de las dos mediciones, por lo que se eligió tomar como pesos $\omega_1 = \omega_2 = 0{,}5$

Estructuras de Datos

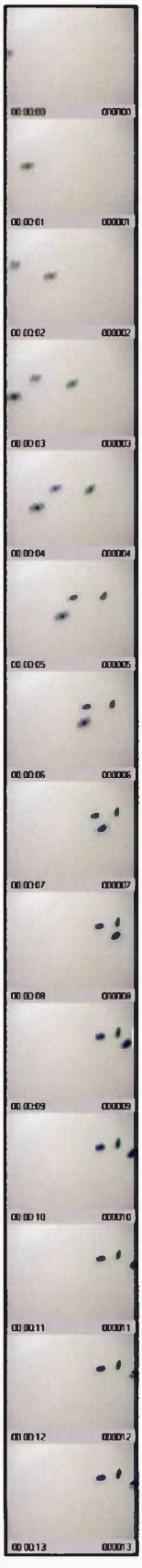
La información acerca de las trayectorias que se van armando se mantiene en un archivo de trayectorias. Para simplificar el algoritmo de intercambio así como para ahorrar tiempo de computación, la información de trayectorias relacionada a los últimos tres cuadros analizados se mantiene siempre en memoria.
La estructura de datos para una trayectoria tiene la siguiente información:
Para cada imagen:

- Cantidad de objetos encontrados en la imagen (**n**)

n veces:

- Posición del objeto

El orden en que quedan grabados los objetos es el que resultó de la aplicación del algoritmo de seguimiento de trayectorias. En el caso de los puntos fantasma como parte de una trayectoria, se guardan también con un valor de fila y columna inválida (-1,-1) lo que lo distingue de un punto común.



EJEMPLO

Sea la secuencia de imágenes de la prueba número 5 (ver APENDICE E: Pruebas realizadas y resultados de la evaluación).

Por la forma en que se desplazan los tres objetos de la escena, el orden en que los mismos son identificados difiere entre los distintos cuadros. Esto implica que a medida que aparecen nuevos cuadros, el procedimiento deberá intercambiar los objetos encontrados con el fin de que las trayectorias encontradas sean las correctas.

En las próximas dos páginas se muestran dos esquemas. El primero muestra el resultado de la identificación de objetos para los 28 cuadros de la secuencia, en el mismo orden en que los mismos fueron identificados. Cada fila representa la información de un cuadro de la secuencia, y cada columna muestra una terna de valores: El área, el color promedio, y la posición de cada objeto encontrado en la imagen.
El segundo esquema despliega la misma información, pero en este caso los objetos ya han sido intercambiados de manera que columna por columna se puede ver el recorrido que ha realizado cada uno de estos objetos.

Nótese que en los primeros cuadros no se muestra información de todos los objetos dado que los mismos no aparecen desde el principio en la escena, sino de a uno por vez.

Imagen	*Area*	*Color*	Posición	*Area*	*Color*	Posición	*Area*	*Color*	Posición
d01.pgm	*131*	*54*	(113, 4)						
d02.pgm	*219*	*54*	(116, 23)						
d03.pgm	*117*	*54*	(118, 54)						
d04.pgm	*57*	*60*	(84, 6)	*84*	*56*	(118, 83)			
d05.pgm	*43*	*61*	(86, 17)	*75*	*58*	(117,112)			
d06.pgm	*72*	*57*	(114,140)	*148*	*62*	(138, 4)			
d07.pgm	*416*	*50*	(141, 18)	*98*	*56*	(109,167)			
d08.pgm	*61*	*58*	(97, 98)	*364*	*53*	(143, 47)	*158*	*56*	(105,188)
d09.pgm	*91*	*53*	(96,123)	*299*	*54*	(143, 76)	*174*	*54*	(99,209)
d10.pgm	*132*	*50*	(94,146)	*294*	*55*	(141,108)	*195*	*52*	(94,227)
d11.pgm	*301*	*54*	(138,138)	*201*	*50*	(89,242)	*164*	*48*	(92,168)
d12.pgm	*341*	*53*	(133,167)	*210*	*47*	(84,254)	*193*	*46*	(89,185)
d13.pgm	*219*	*45*	(81,264)	*221*	*45*	(87,200)	*372*	*51*	(127,194)
d14.pgm	*216*	*43*	(77,271)	*241*	*44*	(86,212)	*379*	*48*	(122,217)
d15.pgm	*211*	*41*	(76,275)	*257*	*43*	(85,222)	*409*	*46*	(116,239)
d16.pgm	*214*	*40*	(74,277)	*261*	*41*	(84,228)	*406*	*44*	(111,257)
d17.pgm	*211*	*40*	(74,277)	*267*	*42*	(84,232)	*414*	*43*	(107,273)
d18.pgm	*211*	*40*	(75,277)	*263*	*41*	(84,234)	*422*	*41*	(104,286)
d19.pgm	*224*	*41*	(75,277)	*273*	*42*	(84,234)	*422*	*39*	(102,298)
d20.pgm	*217*	*39*	(75,278)	*273*	*40*	(85,234)	*357*	*34*	(101,305)
d21.pgm	*218*	*40*	(75,278)	*277*	*42*	(85,235)	*215*	*34*	(102,309)
d22.pgm	*220*	*38*	(76,278)	*279*	*40*	(85,235)	*132*	*33*	(102,310)
d23.pgm	*219*	*39*	(76,278)	*281*	*41*	(85,235)	*104*	*35*	(102,311)
d24.pgm	*220*	*37*	(76,278)	*280*	*39*	(85,235)	*98*	*34*	(102,311)
d25.pgm	*226*	*38*	(76,278)	*285*	*39*	(85,235)	*100*	*33*	(102,311)
d26.pgm	*221*	*37*	(76,278)	*290*	*39*	(85,235)	*97*	*32*	(102,311)
d27.pgm	*220*	*35*	(75,279)	*297*	*39*	(85,235)	*96*	*32*	(102,311)
d28.pgm	*222*	*35*	(75,278)	*299*	*38*	(85,235)	*96*	*30*	(103,311)

a) Identificación de objetos - Disposición original

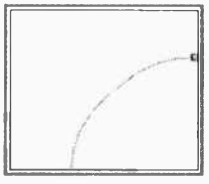
Imagen	Area	Color	Posición	Area	Color	Posición	Area	Color	Posición
d01.pgm	131	54	(113, 4)						
d02.pgm	219	54	(116, 23)						
d03.pgm	117	54	(118, 54)						
d04.pgm	84	56	(118, 83)	57	60	(84, 6)			
d05.pgm	75	58	(117,112)	43	61	(86, 17)			
d06.pgm	72	57	(114,140)	148	62	(138, 4)			
d07.pgm	98	56	(109,167)	416	50	(141, 18)			
d08.pgm	158	56	(105,188)	364	53	(143, 47)	61	58	(97, 98)
d09.pgm	174	54	(99,209)	299	54	(143, 76)	91	53	(96,123)
d10.pgm	195	52	(94,227)	294	55	(141,108)	132	50	(94,146)
d11.pgm	201	50	(89,242)	301	54	(138,138)	164	48	(92,168)
d12.pgm	210	47	(84,254)	341	53	(133,167)	193	46	(89,185)
d13.pgm	219	45	(81,264)	372	51	(127,194)	221	45	(87,200)
d14.pgm	216	43	(77,271)	379	48	(122,217)	241	44	(86,212)
d15.pgm	211	41	(76,275)	409	46	(116,239)	257	43	(85,222)
d16.pgm	214	40	(74,277)	406	44	(111,257)	261	41	(84,228)
d17.pgm	211	40	(74,277)	414	43	(107,273)	267	42	(84,232)
d18.pgm	211	40	(75,277)	422	41	(104,286)	263	41	(84,234)
d19.pgm	224	41	(75,277)	422	39	(102,298)	273	42	(84,234)
d20.pgm	217	39	(75,278)	357	34	(101,305)	273	40	(85,234)
d21.pgm	218	40	(75,278)	215	34	(102,309)	277	42	(85,235)
d22.pgm	220	38	(76,278)	132	33	(102,310)	279	40	(85,235)
d23.pgm	219	39	(76,278)	104	35	(102,311)	281	41	(85,235)
d24.pgm	220	37	(76,278)	98	34	(102,311)	280	39	(85,235)
d25.pgm	226	38	(76,278)	100	33	(102,311)	285	39	(85,235)
d26.pgm	221	37	(76,278)	97	32	(102,311)	290	39	(85,235)
d27.pgm	220	35	(75,279)	96	32	(102,311)	297	39	(85,235)
d28.pgm	222	35	(75,278)	96	30	(103,311)	299	38	(85,235)

b) Identificación de objetos - Disposición según la trayectoria

Representación Visual de las Trayectorias

La información plana (con formato de texto) que constituye las trayectorias encontradas, puede ser representada en forma gráfica. La idea es generar una imagen que para cada objeto identificado, muestre con un trazo el recorrido que el mismo hizo a lo largo de la secuencia de imágenes. El punto inicial de la trayectoria debería destacarse de alguna manera para indicar dónde comienza el movimiento del objeto.

A modo de ejemplo, sea el caso de una secuencia donde aparecen dos pelotas: una rebotando y la otra cayendo desde cierta altura. La imagen de resultados sería similar a la superposición de las siguientes dos imágenes, correspondientes cada una a la trayectoria de cada una de las pelotas:



Si la trayectoria que se quiere describir con una imagen esta incompleta, es decir, tuvo que ser completada con puntos fantasma, podría dejarse en blanco, o podría hacerse una estimación de su posición y proyectar este recorrido dibujándolo con un color de trazo diferente o con una línea punteada.

Esta imagen de visualización de las trayectorias podría armarse una vez analizada toda la secuencia, o podría ir generándose al mismo tiempo en que se realiza el seguimiento.

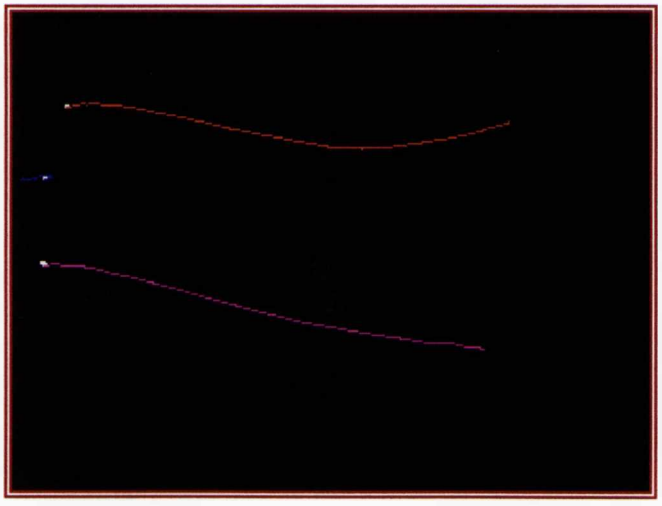
El siguiente algoritmo muestra la idea del armado de las imágenes de trayectorias:

```
Sea MAX la cantidad de objetos en el primer cuadro.
Para cada uno de estos objetos generar una imagen asociada a su trayectoria.
Para cada cuadro
Para I=1 hasta MAX
        Si en el cuadro el punto existe
                Generar una línea desde el punto del objeto en la imagen anterior, hasta
                        el punto del objeto en la imagen actual
        Si en el cuadro el punto no existe (fantasma)
                Proyectar la posición y dibujar en otro color o dejar en blanco
Para cada objeto nuevo que aparezca en el cuadro
        Incrementar MAX en 1
        Crear una nueva imagen de trayectoria.
```

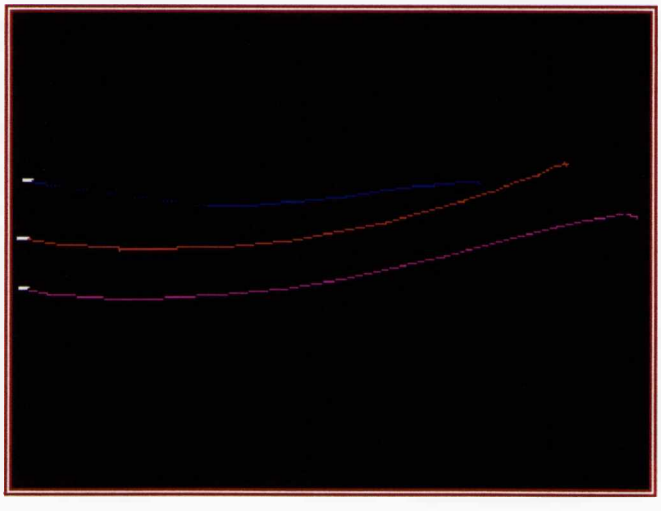
En la presente implementación se creó una aplicación que genera una imagen de trayectorias a partir de la información ya procesada. Cada objeto en movimiento puede distinguirse por un color diferente, mientras que el punto de inicio de los movimientos se destaca en color blanco.
Para el caso de una trayectoria incompleta, la línea del trayecto se completa con una línea punteada en el mismo color.

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

Las figuras (a) y (b) que siguen, muestran las imágenes de trayectoria resultantes de dos de las pruebas descriptas en el APENDICE E: Pruebas realizadas y resultados de la evaluación.



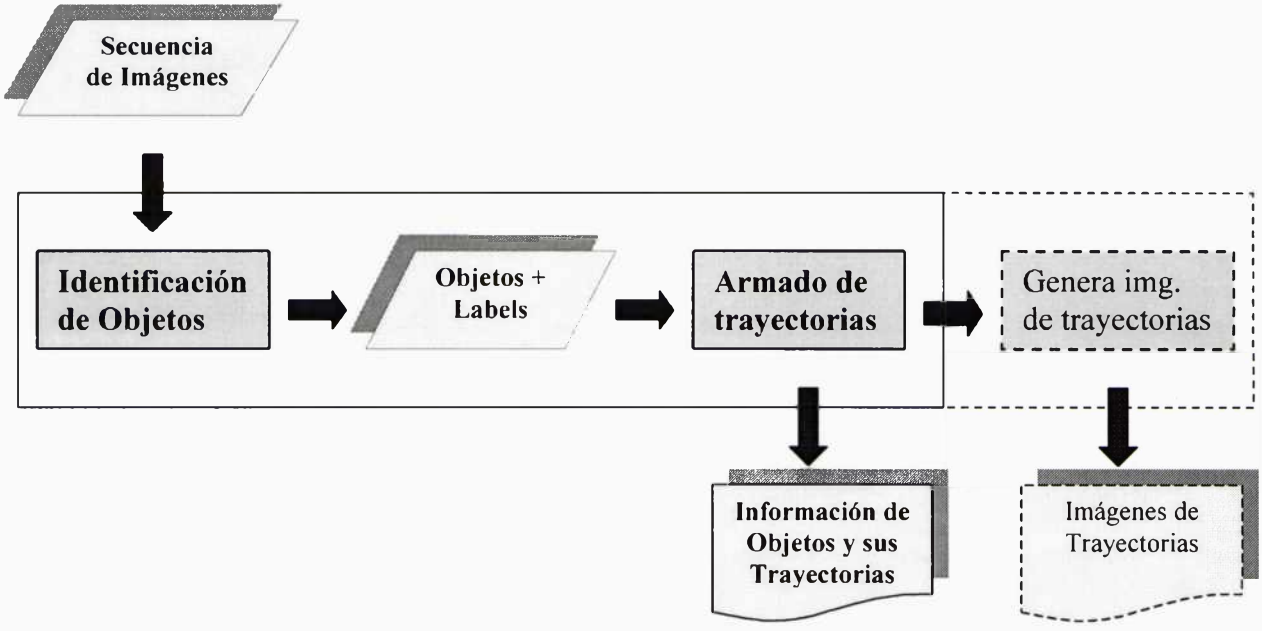
(a) Trayectorias de la prueba nro.3



(b) Trayectorias de la prueba nro.5

Parte III – Integrando la Información

En resumen, se ha llegado a un sistema con las siguientes entradas y salidas:



Varias suposiciones se han hecho al plantear esta solución, y las mismas influyeron directamente en la implementación de los distintos módulos o etapas del sistema. Estas suposiciones son:

- Las imágenes son simples superficies que se destacan sobre un fondo contrastante.
- Los objetos de la escena no se tocan.
- La secuencia de imágenes no se captura en tiempo real.
- Los objetos son indivisibles; no hay características que distingan un objeto de otro.

Por otro lado para cada paso o procedimiento se ha hecho una elección de la técnica utilizada, que no es única, por ejemplo: para realzar una imagen, para identificar un objeto, etc.

Dada la modularidad de la solución, la implementación de cualquiera de los pasos puede ser fácilmente reemplazada o mejorada sin mayor inconveniente. Asimismo, si se quisiera eliminar alguna de las suposiciones anteriores bastará con cambiar una parte acotada de la implementación, puesto que el procedimiento explicado seguirá siendo útil y sirviendo como guía de cualquier manera.

Solución Por Medio De Técnicas De Programación Paralela

Introducción

Ya desarrollada y comprendida la solución tradicional para el problema de reconocimiento de objetos y seguimiento de trayectorias, el siguiente objetivo es la paralelización con el fin principal de mejorar los tiempos de procesamiento teniendo en cuenta la naturaleza del problema en cuestión.
Se planteará a continuación una posible manera de llevar esto a cabo. La situación es la siguiente: Se dispone de varias máquinas conectadas en red, y el programa se ejecuta simultáneamente en todas ellas. En cada máquina corren uno o más procesos, dependiendo de la cantidad de máquinas disponibles y del número de procesos definidos. Usando siempre el mismo programa y con un simple cambio en los parámetros con que se lo invoca, cualquiera de estas dos variables puede ser modificada.
Como ya se mencionó, la implementación de este proyecto se realizó en MPI, por lo tanto en los distintos puntos donde se transcriba un fragmento del programa o se plantee un pseudocódigo, se usarán las primitivas de este lenguaje (para más detalle ver APENDICE A: REFERENCIA DEL LENGUAJE MPI)
Se explicará en primer lugar cómo fue modificada la etapa de reconocimiento de objetos para su ejecución en paralelo.

Parte I – Reconocimiento de Objetos

Realce de las Imágenes

Una buena elección para introducir el paralelismo en el manejo de imágenes es tomar el proceso de realce de una imagen. Retómese la situación original (ver en la solución secuencial, pág. 31). La operación realizada era un simple promedio espacial pesado con una ventana de 5 puntos, combinado con una función de realce del contraste. Como se vio, el ciclo principal que efectúa esta operación sobre la imagen consiste en recorrer todas sus filas y, para cada fila, recorrer todas sus columnas realizando la operación que calcula el punto en la nueva imagen.

Para derivar un código paralelo equivalente, se distribuye entre todos los procesos la matriz que representa la imagen, y a cada uno de ellos se le da la tarea de actualizar la parte de la imagen que le fue asignada. El algoritmo paralelo depende de la elección en la distribución de los datos. Ésta debería ser balanceada, es decir, se debería repartir en la medida de lo posible, el mismo número de puntos a cada uno de los procesos.
La siguiente figura muestra dos posibles distribuciones:

(a) (b)

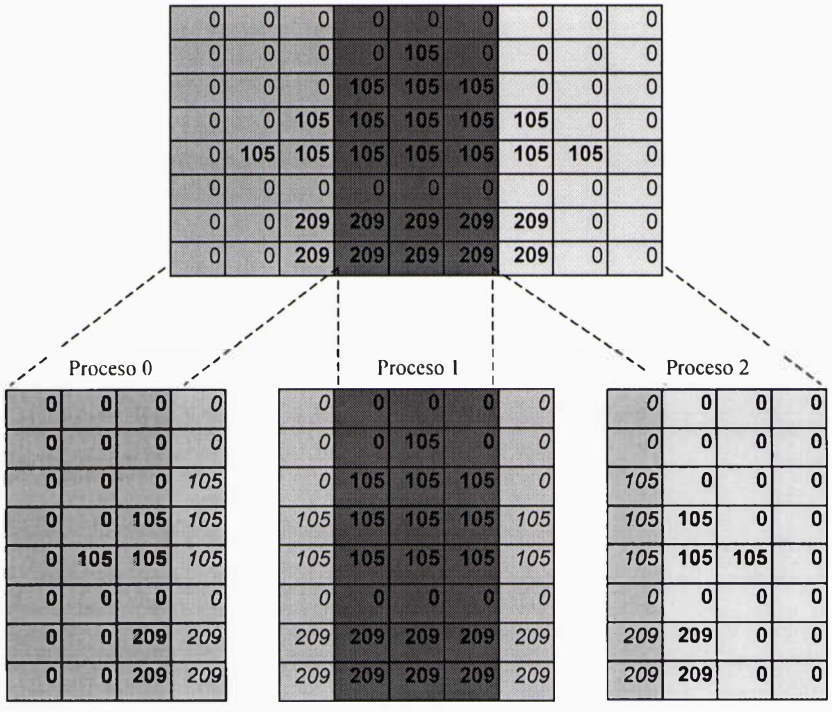
En la primera, la matriz es particionada en grupos de columnas mientras que en la segunda, la imagen se parte en las dos dimensiones. Para tomar la decisión entre estas dos opciones se tuvo en cuenta lo siguiente:

1) Las imágenes no se manejan en memoria sino en archivos y
2) El orden en que se realizan la lectura de la imagen original y la grabación de las imágenes modificadas depende directamente de la forma en que las imágenes están grabadas.

Con estos puntos en mente, es claro que la opción adecuada es la primera ya que a medida que se recorre la imagen original todos los procesos pueden ir generando la imagen modificada.

Como se vio, el valor de cada pixel en la nueva imagen se computa a partir del valor de los puntos que "caen" dentro de la ventana que se está utilizando. En consecuencia, las fronteras de las subimágenes - en este caso la primer y la última columna de cada proceso - no pueden procesarse adecuadamente sin conocer datos de las columnas adyacentes. Estas columnas han sido asignadas a otros procesos, por lo que resulta inevitable que esos datos se compartan. Esto implica que al repartir los datos, entre cada par de procesos haya una superposición de una columna de datos.

Para ver esas ideas más claramente sea el ejemplo visto bajo "Reconocimiento de Objetos" (pág. 35), y supóngase que el número de procesos participantes de la operación es tres. La manera de repartir los datos de la imagen sería darle tres columnas a cada proceso.



En el gráfico se ve qué columnas le corresponden a cada proceso y además, en letra itálica, la columna adicional que cada uno debe ver para poder realizar su tarea.

Ya definida qué parte de la operación debe hacer cada proceso, falta todavía establecer:
a) ¿De dónde obtiene los datos cada proceso?
b) ¿Dónde quedan los resultados?

Considérese la obtención de los datos. Una posibilidad es suponer que inicialmente la imagen está replicada tantas veces como procesos hay, por lo que todos los procesos ven la totalidad de la información. Así, cada uno después de calcular cuál es la parte de la imagen que le corresponde toma solamente la información que necesita.
Sea ahora la decisión sobre dónde dejar la imagen resultante. Para que un proceso vea esta nueva imagen en forma completa, debería recopilar los resultados producidos por todos los demás y combinarlos junto con los datos propios para armar la imagen final (ver Figura A en la siguiente página).

¿Cómo hace cada proceso para calcular la parte de la imagen sobre la que tiene que operar? Como se vio anteriormente, todo lenguaje de programación paralelo - y en particular MPI - provee por lo menos el siguiente par de funciones: una para determinar cuántos procesos componen el total, y otra para conocer su número o identificación dentro del conjunto. Además, como se dispone de la imagen completa, cada proceso puede averiguar, leyendo los datos de la imagen, la dimensión de la misma, es decir, su número de filas y de columnas. A partir de estos datos el cálculo es directo: Siendo *cantProc* la cantidad de procesos, *i* el número de proceso, y *M* la cantidad de columnas de la imagen, la cantidad de columnas del proceso *i* se calcula de la siguiente manera:

$$CantColumnas_i = M / cantProc$$

$$Si\ i < M - cantProc * cantColumnas_i\ entonces\ cantColumnas_i = cantColumnas_i + 1$$

La segunda línea asegura que todas las columnas son repartidas; sirve en el caso que la cantidad de columnas no sea múltiplo exacto de la cantidad de procesos.

Apuntando a un sistema de procesamiento en tiempo real, no es aceptable la replicación de las imágenes como se planteó acá. Por el contrario, se debe tratar que el flujo de los datos sea el menor posible. Una solución entonces es designar un proceso como el único con acceso a la imagen original, y que ese proceso sea el responsable de tomar la imagen - en este caso acceder al disco para tomar el archivo correspondiente -, leer la información de la misma y repartirla entre los procesos restantes. De esta manera, la única información que "viajaría" a través de la red sería exactamente la que cada proceso necesita, incluyendo por supuesto tanto las columnas principales que debe procesar como las adyacentes. Como se explicó, cada proceso necesita conocer además de la información que procesa, las dimensiones de la imagen. Este es un dato extra que el proceso maestro le debe enviar a cada uno. Este proceso especial puede o no ser uno de los que realiza la tarea principal de realce.
En esta implementación se decidió que todos los procesos participen de la tarea, y en particular, que un proceso – el llamado raíz o "*root*" – haga además esta tarea de interfaz entre los procesos y las imágenes.

Por otra parte el hecho de que todos los procesos envíen los resultados obtenidos a cada uno de los demás procesos, generaría un tráfico desmedido de información salvo que realmente cada proceso necesite tener este resultado. Entonces, así como se definió al proceso raíz como

encargado de tomar y repartir la imagen original, se designó al mismo proceso como el que recopile todos los resultados y "arme" la imagen final. (ver Figura B).

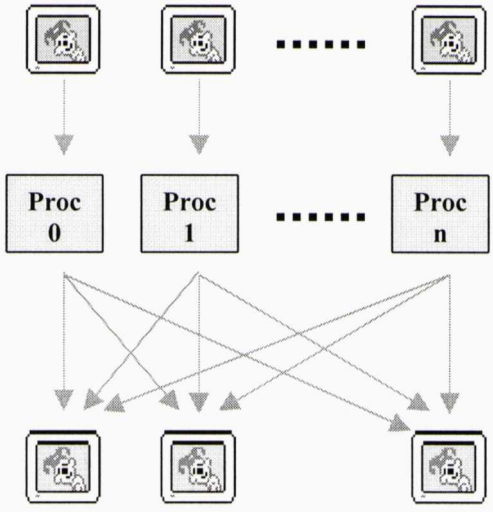


Figura A

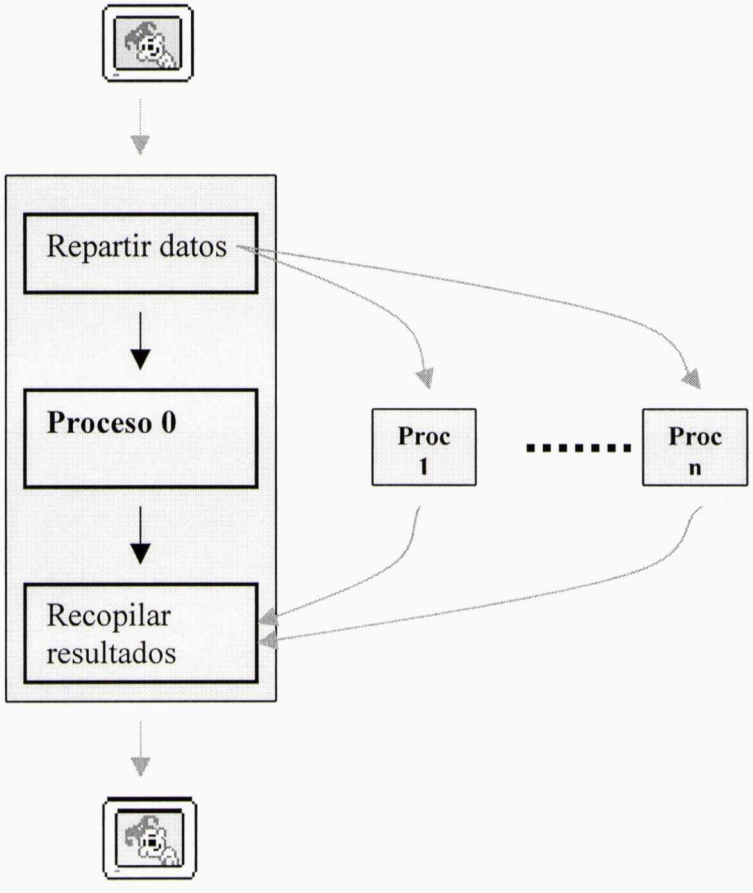


Figura B

Como derivación de todas estas consideraciones, el proceso paralelizado para realzar una imagen queda de la siguiente manera: Cada proceso recibe una fila de la imagen que el proceso maestro le envía, genera la información para la nueva imagen, le envía los resultados al proceso maestro y espera la información de la fila siguiente. El proceso maestro por su parte, lee una fila de la imagen, reparte la información entre los procesos restantes – quedándose por supuesto con la parte que le corresponde a sí mismo -, realiza la parte de la tarea que le toca mientras los procesos restantes hacen su parte, y por último recopila los resultados dejándolos en la nueva imagen. Toma otra fila de la imagen original y repite los pasos anteriores hasta haber procesado toda la información. Otra elección hubiese sido repartir toda la imagen, esperar que cada proceso haga toda su tarea, y por último, recopilar todos los resultados en el nodo correspondiente al proceso maestro o raíz. Se optó por la primera por el motivo ya mencionado anteriormente, que es el hecho de que las imágenes están grabadas en archivo y en consecuencia resulta ésta la mejor manera de manipular la información.
A continuación se muestra el algoritmo que implementa estos conceptos:

Cuerpo principal del proceso

```
Root= 0                 // se decide que el proceso "root" sea el número 0

MPI_INIT                // Inicialización de MPI

MPI_COMM_rank(MPI_COMM_WORLD, mirank)       // averigua el número de proceso ( 0 a n-1)

Si mirank = root
        Master          // Código que debe ejecutar el proceso raíz
Si no
        Slave           // Código que deben ejecutar los procesos restantes


MPI_Finalize            // Fin de la ejecución
```

Master (Código que ejecuta el proceso raíz)

```
MPI_COMM_size(MPI_COMM_WORLD, cantProc)     // averigua el número de procesos

Obtener el número de filas (N) y de columnas (M) de la imagen

MPI_Broadcast(M, ..., root, MPI_COMM_WORLD)     // Envía la información a todos los procesos
MPI_Broadcast(N, ..., root, MPI_COMM_WORLD)


// computa el número de columnas que le toca a cada proceso
Para i=1 hasta cantProc
        CantColumnas i = M / cantProc
        Si i < M – cantProc * cantColumnas i entonces cantColumnas i = cantColumnas i + 1

Crear la nueva imagen sin datos

Para i=1 hasta 2 {
        Obtener la fila i de la imagen

        // Reparte la información entre los procesos
```

```
        Para p=1 hasta cantProc-2
                MPI_Send(datos de la fila i, CantColumnas p+2, ..., p, ...)      // Envía al proceso p

        MPI_Send(datos de la fila i, CantColumnas p+1, ..., CantProc-1, ...)     // Envía al
                                                                                 // último proceso
        }

// Graba la fila 1 en la nueva imagen (sin cambios)
Para j=1 hasta M
        B(1, j) = A(1, j)

Para i=3 hasta N {

        Obtener la fila i de la imagen

        // Reparte la información entre los procesos
        Para p=1 hasta cantProc-2
                MPI_Send(datos de la fila i, CantColumnas p+2, ..., p, ...)      // Envía al proceso p

        MPI_Send(datos de la fila i, CantColumnas p+1, ..., CantProc-1, ...)     // Envía al
                                                                                 // último proceso

        // Procesa la parte que le corresponde de la fila i - 1
        k = i – 1

        Para j=2 hasta cantColumnas root
                B(k,j)=1/8 A(k-1, j-1) + 1/8 A(k-1, j) + 1/4 A(k, j) + 1/8 A(k+1,j) + 1/8 A(k, j+1)

        // Recopila la información generada por los procesos
        Para p=1 hasta cantProc
                MPI_Recv(datos para la fila i - 1, CantColumnas p, ... , p, ...)  // Recibe del proc. p

        Grabar fila i-1 en la nueva imagen

        }
// Graba la fila N en la nueva imagen (sin cambios)
Para j=1 hasta M
        B(n, j) = A(n, j)
```

Slave (Código que ejecutan todos los procesos excepto el proceso raíz)

```
MPI_Broadcast(M, ..., root, MPI_COMM_WORLD)      // Recibe de "root" las dimensiones de
MPI_Broadcast(N, ..., root, MPI_COMM_WORLD)      // la imagen

// computa el número de columnas que le toca
CantColumnas mirank = M / cantProc
Si i < M – cantProc * cantColumnas mirank entonces cantColumnas mirank = cantColumnas mirank + 1

// Recibe de "root" las dos primeras filas
Para i=1 hasta 2 {
        Si mirank = cantProc - 1
                MPI_Recv(datos de la fila i, CantColumnas mirank+1, ..., root, ...) // Recibe de "root"
        Si no
                MPI_Recv(datos de la fila i, CantColumnas mirank+2, ..., root, ...) // Recibe de "root"
                }

Para i=3 hasta N {
```

```
Si mirank = cantProc - 1
        MPI_Recv(datos de la fila i, CantColumnas mirank+1, ..., root, ...) // Recibe de "root"
Si no
        MPI_Recv(datos de la fila i, CantColumnas mirank+2, ..., root, ...) // Recibe de "root"

// Procesa la parte que le corresponde para la fila i – 1
k = i - 1
Para j=1 hasta cantColumnas mirank
        B(k,j)=1/8 A(k, j-1) + 1/8 A(k-1, j) + 1/4 A(k, j) + 1/8 A(k+1,j) + 1/8 A(k, j+1)

MPI_Send(datos para la fila i - 1, CantColumnas mirank, ... , root, ...)      // Envía a "root"
                                                                              // el resultado
}
```

El algoritmo da lugar a un par de comentarios:

- Notar que el proceso raíz envía a cada proceso CantColumnas $_i$ +2 columnas de datos, es decir, le envía las columnas que debe procesar más las dos columnas adyacentes que le sirven para poder realizar la operación. La excepción son los dos procesos de los bordes de la imagen. Uno de ellos es él mismo y el otro es el último proceso. Por lo tanto a éste le envía sólo una columna de más que es la que está a la izquierda de las que debe procesar. Al recopilar los datos resultantes, el proceso raíz recibe por supuesto solamente la cantidad de columnas exactas que cada uno procesó.
- El proceso raíz envía primero las dos primeras filas y recién a partir de la tercera los procesos comienzan a operar sobre los datos. Esto es simplemente porque de otra manera los mismos no dispondrían de toda la información para realizar los cálculos.

Reconocimiento de Objetos

El proceso de reconocimiento de objetos tal como se vio en la primera solución, se simplificó a una tarea de **thresholding** (para separar la zona de objetos de la zona del fondo) en conjunto con un procedimiento de **labeling** (para separar los distintos objetos de la escena). La paralelización de este procedimiento se basa en la misma idea pero introduce algunos detalles que ahora se verán.

En principio la aplicación de paralelismo en este algoritmo es similar a la del procedimiento de realce de imágenes. También en este caso se tiene al proceso "raíz" haciendo las veces de intermediario entre la imagen y los procesos restantes. Este proceso reparte las columnas de la imagen en forma equitativa entre los distintos procesos y luego cada uno realiza una parte de la operación general. La parte de la tarea que le corresponde a cada proceso consiste en recorrer la subimagen que se le asignó tratando de extraer de ella toda la información relativa a los objetos candidatos. Esto implica realizar los mismos pasos de segmentación e identificación de objetos vistos anteriormente. Para resumir la operación, todos los procesos envían la información obtenida al proceso raíz de modo que el resultado del proceso de reconocimiento de objetos en la imagen queda centralizado en un único nodo.

El siguiente algoritmo muestra cómo se desarrolla la tarea de reconocimiento en forma paralela para el proceso master en primer lugar, y para los restantes después:

Master (Código que ejecuta el proceso raíz)

```
MPI_COMM_size(MPI_COMM_WORLD, cantProc)      // averigua el número de procesos

Obtener el número de filas (N) y de columnas (M) de la imagen

MPI_Broadcast(M, ..., root, MPI_COMM_WORLD)      // Envía la información a todos los procesos
MPI_Broadcast(N, ..., root, MPI_COMM_WORLD)


// computa el número de columnas que le toca a cada proceso
Para i=1 hasta cantProc
        CantColumnas i = M / cantProc
        Si i < M – cantProc * cantColumnas i entonces cantColumnas i = cantColumnas i + 1


Para i=1 hasta N {

        Obtener la fila i de la imagen

        // Reparte la información entre los procesos
        Para p=1 hasta cantProc-2
                MPI_Send(datos de la fila i, CantColumnas p+2, ..., p, ...)      // Envía al proceso p

        MPI_Send(datos de la fila i, CantColumnas p+1, ..., CantProc-1, ...)     // Envía al
                                                                                 // último proceso

        // Procesa la parte que le corresponde de la fila i
```

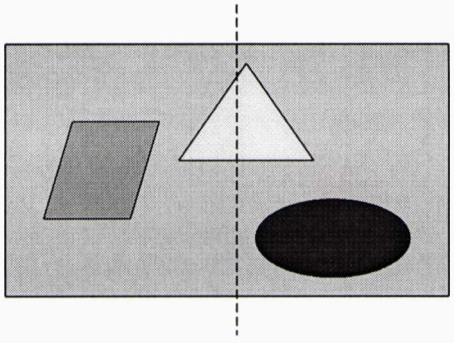
```
        Para j=2 hasta cantColumnas root
                {
                tomar el punto Pij
                comparar su valor de intensidad de gris contra t
                si es mayor
                        analizar la conectividad con sus vecinos para ver a qué objeto pertenece
                                si pertenece a un único objeto ya encontrado
                                        colocar la etiqueta correspondiente a ese objeto
                                        actualizar los datos del objeto con los datos del punto Pij
                                si no, si pertenece a varios objetos candidatos
                                        combinar la información de los objetos, dejando uno solo
                                        colocar la etiqueta correspondiente a ese objeto
                                        actualizar los datos del objeto con los datos del punto Pij
                                si no
                                        asociarle una etiqueta nueva
                                        crear un objeto nuevo con los datos del punto Pij
                }
        Para cada objeto encontrado
                Si no cumple con  los requisitos predeterminados para ser un objeto de interés
                        Descartar el objeto

        // Recopila la información de objetos encontrados por los procesos
        Para p=1 hasta cantProc
                MPI_Recv(<info obj.>, ... , p, ...)// Recibe del proc. p

        }
```

Slave (Código que ejecutan todos los procesos excepto el proceso raíz)

```
MPI_Broadcast(M, ..., root, MPI_COMM_WORLD)      // Recibe de "root" las dimensiones de
MPI_Broadcast(N, ..., root, MPI_COMM_WORLD)      // la imagen

// computa el número de columnas que le toca
CantColumnas mirank = M / cantProc
Si i < M – cantProc * cantColumnas mirank entonces cantColumnas mirank = cantColumnas mirank + 1


Para i=1 hasta N {

        Si mirank = cantProc - 1
                MPI_Recv(datos de la fila i, CantColumnas mirank+1, ..., root, ...) // Recibe de "root"
        Si no
                MPI_Recv(datos de la fila i, CantColumnas mirank+2, ..., root, ...) // Recibe de "root"

        // Procesa la parte que le corresponde de la fila i

        Para j=2 hasta cantColumnas mirank
                {
                tomar el punto Pij
                comparar su valor de intensidad de gris contra t
                si es mayor
                        analizar la conectividad con sus vecinos para ver a qué objeto pertenece
                                si pertenece a un único objeto ya encontrado
                                        colocar la etiqueta correspondiente a ese objeto
                                        actualizar los datos del objeto con los datos del punto Pij
                                si no, si pertenece a varios objetos candidatos
                                        combinar la información de los objetos, dejando uno solo
```

```
                                        colocar la etiqueta correspondiente a ese objeto
                                        actualizar los datos del objeto con los datos del punto Pij
                                si no
                                        asociarle una etiqueta nueva
                                        crear un objeto nuevo con los datos del punto Pij
        }
Para cada objeto encontrado
        Si no cumple con  los requisitos predeterminados para ser un objeto de interés
                Descartar el objeto


MPI_Send(<info obj.>, ... , root, ...)        // Envía a "root" la info. de objetos encontrados
}
```

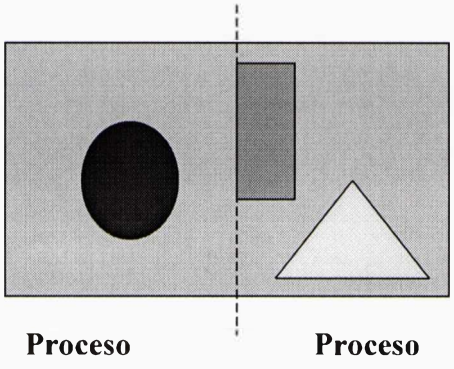
En el mejor de los casos, al finalizar la tarea cada proceso obtendrá como resultado un subconjunto de los objetos de la imagen. La suma de todos estos subconjuntos será el conjunto total de los objetos que se encuentran en la imagen.
Pero esta solución tiene un problema. A veces la suma de todos los resultados es un resultado parcial ya que es posible que uno o más objetos hayan quedado "partidos" a lo largo de dos o más subimágenes, es decir, que no van a ser completamente identificados por un único proceso. El ejemplo dado al principio es una muestra clara de esta situación. En la imagen hay un triángulo que ninguno de los tres procesos va a reconocer; cada uno verá sólo una parte del mismo.
La solución que se encontró para estos casos fue que cada proceso identifique entre todos los objetos candidatos encontrados aquellos que están "pegados" al borde izquierdo. Esto es una indicación bastante certera de que se trata de un objeto que continúa en la subimagen adyacente. La excepción por supuesto es el caso del proceso que posee la primera subimagen, la cual no linda con ninguna otra. En esta situación, el proceso envía la información relacionada con el objeto al proceso que tiene la tarea de procesar la subimagen de la izquierda. A partir de ese momento "se olvida" que encontró ese objeto; se lo delega al otro proceso y lo descarta de su propio conjunto de objetos.
Sea el punto de vista del otro proceso, el que tiene la subimagen de la izquierda. Después de la primera parte del procedimiento, este proceso obtuvo, como cada uno de los otros, un determinado conjunto de objetos. Probablemente alguno de estos objetos está sobre el borde derecho de la subimagen. La idea entonces es que al recibir la información del proceso vecino verifique si coincide con la frontera de alguno de estos objetos lindantes con el borde derecho.
En el caso que haya coincidencia, el proceso unifica la información, sumando la del objeto "heredado" con la del objeto propio.
El siguiente es un ejemplo muy sencillo de la situación planteada:



Proceso **Proceso**

En este ejemplo el **proceso B** encuentra la elipse, a la cual considera un objeto propio, y el medio triángulo, que al estar sobre su margen izquierdo, lo envía al **proceso A**. A su vez, el **proceso A** encuentra el paralelogramo, y el otro medio triángulo que luego de recibir la información del **proceso B**, reconstruye para obtener la información del triángulo completo.

En los bordes de las subimágenes no siempre va a haber coincidencia. Es el caso por ejemplo de la siguiente figura:



En este caso el proceso **B** envía la información del rectángulo encontrado al proceso **A**, pero resulta que éste no coincide con ninguno de los objetos propios de **A**. Si es este el caso el proceso que recibe este objeto lo agrega como un objeto nuevo. Es claro que ésta no va a ser la situación más común, ya que es muy poco probable que un objeto en una imagen tenga un borde vertical perfecto, y además que éste coincida exactamente con el borde de una subimagen.

Volviendo al algoritmo principal, queda pendiente ver cómo termina. Recién cuando se han resuelto todos los temas de objetos "cortados", cada proceso puede decidir descartar aquellos objetos que no son relevantes (por ejemplo porque el área que ocupa no alcanza el mínimo establecido para que sea considerado como un objeto). Por último cada uno de los procesos envía al proceso maestro la información final de los objetos encontrados, con lo cual finaliza el procedimiento.

Todas estas ideas se reflejan en el algoritmo mejorado:

Master (Código que ejecuta el proceso raíz)

```
MPI_COMM_size(MPI_COMM_WORLD, cantProc)      // averigua el número de procesos

Obtener el número de filas (N) y de columnas (M) de la imagen

MPI_Broadcast(M, ..., root, MPI_COMM_WORLD)      // Envía la información a todos los procesos
MPI_Broadcast(N, ..., root, MPI_COMM_WORLD)
```

```
// computa el número de columnas que le toca a cada proceso
Para i=1 hasta cantProc
        CantColumnas i = M / cantProc
        Si i < M – cantProc * cantColumnas i entonces cantColumnas i = cantColumnas i + 1


Para i=1 hasta N {

        Obtener la fila i de la imagen

        // Reparte la información entre los procesos
        Para p=1 hasta cantProc-2
                MPI_Send(datos de la fila i, CantColumnas p+2, ..., p, ...)        // Envía al proceso p

        MPI_Send(datos de la fila i, CantColumnas p+1, ..., CantProc-1, ...)       // Envía al
                                                                                   // último proceso

        // Procesa la parte que le corresponde de la fila i

        Para j=2 hasta cantColumnas root
                {
                tomar el punto Pij
                comparar su valor de intensidad de gris contra t
                si es mayor
                        analizar la conectividad con sus vecinos para ver a qué objeto pertenece
                                si pertenece a un único objeto ya encontrado
                                        colocar la etiqueta correspondiente a ese objeto
                                        actualizar los datos del objeto con los datos del punto Pij
                                si no, si pertenece a varios objetos candidatos
                                        combinar la información de los objetos, dejando uno solo
                                        colocar la etiqueta correspondiente a ese objeto
                                        actualizar los datos del objeto con los datos del punto Pij
                                si no
                                        asociarle una etiqueta nueva
                                        crear un objeto nuevo con los datos del punto Pij
                }


        // Recopila la información de objetos encontrados sobre el borde
        // por el proceso de su derecha
        Mientras proc.1 tiene objetos sobre su borde izquierdo
                MPI_Recv(<info obj.>, ... , 1, ...)          // Recibe del proc. 1 un objeto
                Si coincide con un objeto propio sobre la derecha
                        Unificar
                Si no
                        Agregar nuevo objeto

        Para cada objeto
                Si no cumple con  los requisitos predeterminados para ser un objeto de interés
                        Descartar el objeto

        // Recopila la información de objetos "enteros" encontrados por los procesos
        Para p=1 hasta cantProc
                MPI_Recv(<info obj.>, ... , p, ...)// Recibe del proc. p


        }
```

Slave (Código que ejecutan todos los procesos excepto el proceso raíz)

```
MPI_Broadcast(M, ..., root, MPI_COMM_WORLD)      // Recibe de "root" las dimensiones de
MPI_Broadcast(N, ..., root, MPI_COMM_WORLD)      // la imagen

// computa el número de columnas que le toca
CantColumnas mirank = M / cantProc
Si i < M – cantProc * cantColumnas mirank entonces cantColumnas mirank = cantColumnas mirank + 1


Para i=1 hasta N {

        Si mirank = cantProc - 1
                MPI_Recv(datos de la fila i, CantColumnas mirank+1, ..., root, ...) // Recibe de "root"
        Si no
                MPI_Recv(datos de la fila i, CantColumnas mirank+2, ..., root, ...) // Recibe de "root"

        // Procesa la parte que le corresponde de la fila i
        Para j=2 hasta cantColumnas mirank
                {
                tomar el punto Pij
                comparar su valor de intensidad de gris contra t
                si es mayor
                        analizar la conectividad con sus vecinos para ver a qué objeto pertenece
                                si pertenece a un único objeto ya encontrado
                                        colocar la etiqueta correspondiente a ese objeto
                                        actualizar los datos del objeto con los datos del punto Pij
                                si no, si pertenece a varios objetos candidatos
                                        combinar la información de los objetos, dejando uno solo
                                        colocar la etiqueta correspondiente a ese objeto
                                        actualizar los datos del objeto con los datos del punto Pij
                                si no
                                        asociarle una etiqueta nueva
                                        crear un objeto nuevo con los datos del punto Pij
                }

        // Recopila la información de objetos encontrados sobre el borde
        // por el proceso de su derecha
        Si mirank< cantProc – 1
                Mientras proc.mirank+1 tiene objetos sobre su borde izquierdo
                MPI_Recv(<info obj.>, ... , mirank+1, ...)      // Recibe del proc. mirank+1 un objeto
                Si coincide con un objeto propio sobre la derecha
                        Unificar
                Si no
                        Agregar nuevo objeto

        Para cada objeto
                Si está sobre su borde izquierdo
                        MPI_Send(<info obj.>, ... , mirank-1, ...)      // Envía al proc. mirank-1 un objeto
                Si no
                        Si no cumple con  los requisitos predeterminados para ser un obj. de interés
                                Descartar el objeto

        // Envía al proceso master la información de objetos "enteros" encontrados
        MPI_Send(<info objetos>, ... , root, ...)  // Envía al proceso root
```

Con respecto a la unificación de las partes de objetos, hay que notar que es necesario que en primer lugar se reciba la información de un proceso – en este caso del de la derecha - y después se envíe la información propia a otro – el de la izquierda. Para entenderlo basta retomar el ejemplo de las figuras. El proceso del centro recibe la porción del triángulo que encontró el de la derecha, y la une con la parte del triángulo que el mismo encontró. Luego envía este objeto – unión de las dos partes – al primer proceso, el cual unifica todo y obtiene como resultado la representación completa del triángulo original. ¿Qué pasaría si el orden fuera inverso? El proceso central enviaría la información que tiene al proceso de la izquierda. Aquél uniría esa información con la propia, obteniendo una figura que no es el triángulo de la imagen. Nuevamente el proceso del centro recibe el pedacito de triángulo del tercer proceso, y lo considera un objeto nuevo. Como resultado no se obtiene la figura original, sino dos figuras separadas.

Estructuras de Datos

Como en la solución secuencial, también en ésta la información de objetos y etiquetas se mantiene en archivos. Salvo por un pequeño agregado, éstos conservan la estructura original.
Cada proceso maneja sus propios archivos de etiquetas y de objetos. El de etiquetas corresponde a la parte de la subimagen que cada uno procesa. Luego de finalizado el proceso de reconocimiento de objetos, el archivo de objetos, contiene:
- toda la información de los objetos que encuentra, más
- la información que agrega en el caso que unifique la propia con la enviada por el proceso de la derecha,

y no tiene:
- aquella información que encontró originalmente, pero que después delegó al proceso de la izquierda.

Como excepción está el proceso raíz, que, como se vio, al terminar el proceso tiene toda la información de todos los objetos de la imagen.

El agregado que se mencionó está en la estructura del archivo de objetos, en la cual aparecen dos datos nuevos:
- Borde izquierdo: Indica si el objeto en cuestión limita a la izquierda
- Borde derecho: Indica si el objeto en cuestión limita a la derecha

Como se puede adivinar, estos nuevos datos sirven para la segunda parte del procedimiento. Cuando los procesos colaboran entre sí para el armado de los objetos que quedaron "partidos", usan estos datos como banderas para determinar si un objeto "cae" o no sobre el borde. Para completar la operación también se valen del archivo de etiquetas que les permite resolver cuándo dos objetos coinciden o no en los bordes.

Selección del Threshold

Como se vio en la solución secuencial, para la selección del threshold se eligió un método iterativo. Este método comienza con la selección de un valor estimado y continúa con un ciclo que mejora el valor del threshold y que se repite hasta que ese valor se estabiliza.

Sea la primer parte: La selección del valor inicial. Igual que antes, se toma el valor medio de los puntos de la imagen. Un cálculo secuencial de este número – ya visto en la solución secuencial - es tan simple como lo muestra este cuadro:

```
Total = 0
Para i = 1 hasta n
        Para j = 1 hasta m
                Total = Total + A (i, j)

Threshold = Total / ( n * m )
```

Para la resolución paralela de este cálculo se resolvió dividir las columnas de la imagen entre los distintos procesos de la misma manera que se hizo para el caso del realce de imágenes. Cada proceso entonces se ocupa de sumar los valores de todos los puntos de las columnas de la imagen que le fueron asignadas. Luego, entre todos realizan una suma en conjunto. Como resultado de esta suma, en cada nodo queda el total calculado equivalente al valor "Total" del algoritmo secuencial. Por último cada proceso realiza la división y así cada uno obtiene como resultado el valor que se esperaba. Como en los casos anteriores también existe el proceso raíz que reparte la información entre los demás procesos. El algoritmo queda como sigue:

Cuerpo principal del proceso

```
MPI_COMM_rank(MPI_COMM_WORLD, mirank)        // averigua el número de proceso ( 0 a n-1)

Si mirank = root
        Master          // Código que debe ejecutar el proceso raíz
Si no
        Slave           // Código que deben ejecutar los procesos restantes
```

Master (Código que ejecuta el proceso raíz)

```
. . .
Averigua el número de procesos
Obtiene el número de filas (n) y de columnas (m) de la imagen y la envía a todos los procesos
Calcula el número de columnas que le toca a cada proceso
. . .

MiSuma = 0

Para i=1 hasta N {
        Obtener la fila i de la imagen

        // Reparte la información entre los procesos
        Para p=1 hasta cantProc-1
                MPI_Send(datos de la fila i, CantColumnas p, ..., p, ...)  // Envía al proceso p
```

```
        // Suma las columnas que le corresponden de la fila i
        Para j=1 hasta cantColumnas root
                MiSuma = MiSuma + A(i, j)
        }
MPI_AllReduce( MiSuma, Total, .... , MPI_SUM, ....)
Threshold = Total / (n * m)
```

Slave (Código que ejecutan todos los procesos excepto el proceso raíz)

```
. . .
Recibe de "root" las dimensiones de la imagen (n, m)
Computa el número de columnas que le toca procesar
. . .

MiSuma = 0

Para i=1 hasta N {
        // Recibe de "root" la información de  la fila i
        MPI_Recv(datos de la fila i, CantColumnas mirank, ..., root, ...)

        // Suma las columnas que le corresponden de la fila i
        Para j=1 hasta cantColumnas mirank
                MiSuma = MiSuma + A(i, j)
        }

MPI_AllReduce( MiSuma, Total, .... , MPI_SUM, ....)
Threshold = Total / (n * m)
```

En esta primer parte del algoritmo para la selección del threshold se introdujo una operación de reducción global: MPI_AllReduce. En este caso todos los procesos colaboran entre sí para realizar la suma de sus respectivas sumas parciales. El tipo de operación a realizar es indicado con la constante **MPI_SUM** que indica "sumar" y corresponde a una de las operaciones predefinidas provistas por MPI. Los valores que se suman son los correspondientes a *MiSuma* y el total queda almacenado en la variable *Total*. En este caso, el totoal calculado queda disponible en cada uno de los procesos que participan de la operación. Esto difiere de la versión más simple MPI_Reduce introducida al explicar los fundamentos de MPI, donde el resultado de la operación queda sólo disponible en el nodo raíz.

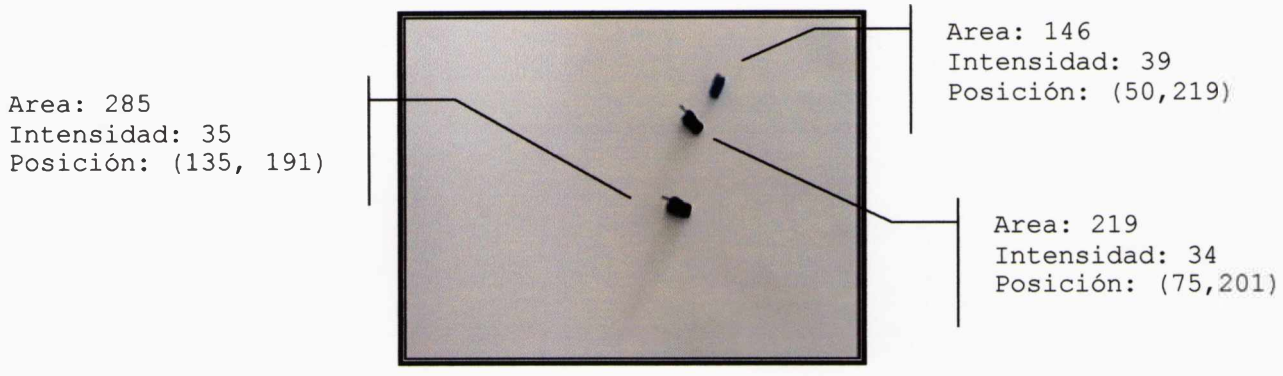
Una vez obtenido el valor inicial, falta realizar el ciclo para la búsqueda del valor definitivo. La idea del algoritmo es dividir la imagen en dos regiones dependiendo de si los valores de los puntos están por arriba o por debajo del threshold inicial. El ciclo se repite hasta que en dos iteraciones sucesivas el valor permanezca sin cambios. La paralelización de este algoritmo es bastante similar a la anterior. Los procesos se reparten las columnas y calculan las sumas parciales, en esta caso, correspondientes a cada una de las dos regiones. Luego, mediante una operación conjunta calculan las dos medias. Finalmente cada uno hace el cálculo del nuevo valor para el threshold y decide si debe repetir o no el ciclo.
El algoritmo secuencial queda modificado como sigue:

Master (Código que ejecuta el proceso raíz)

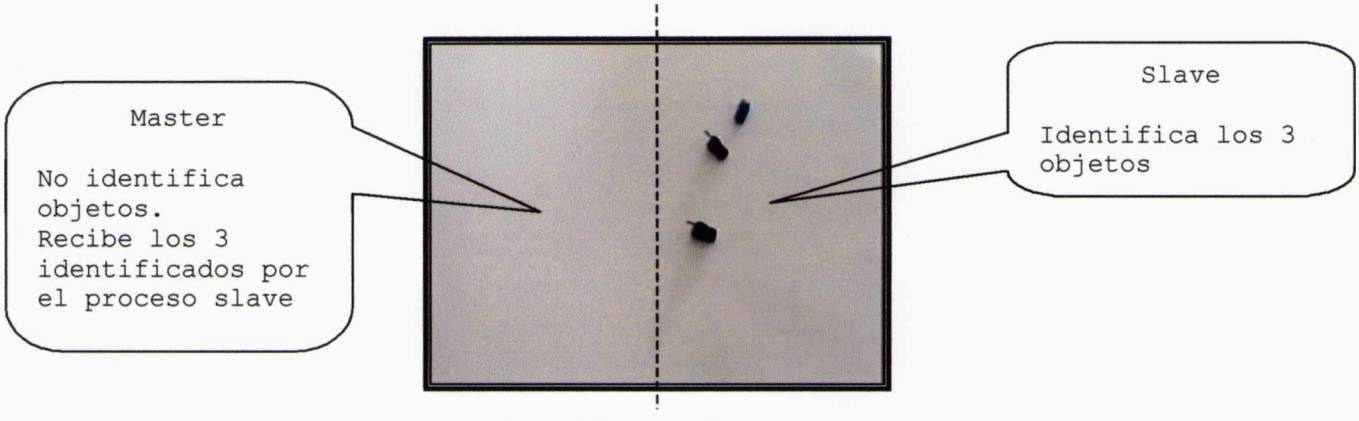
```
/*  Variables:
    m1Anterior, m2Anterior: los valores medios de las regiones en el ciclo anterior.
    threshold: el valor del threshold que se esta calculando
    R1, R2: la suma de los valores de los puntos en las dos regiones
    cantPuntos1, cantPuntos2: La cantidad de puntos en las regiones R1 y R2 respectivamente
    A(i, j): El valor del punto de la imagen en la posición (i, j)

m1Anterior = m2Anterior = 0
threshold = m1 = m2 = *media de la imagen*
Mientras (m1Anterior <> m1 y m2Anterior <> m2)
        {
        R1 = R2 = cantPuntos1 = cantPuntos2 = 0
        Para i=1 hasta n
                {
                Obtener la fila i de la imagen

                // Reparte la información entre los procesos
                Para p=1 hasta cantProc-1
                        MPI_Send(datos de la fila i, CantColumnas p, ..., p, ...)  // Envía al proceso p

                // Procesa las columnas que le corresponden de la fila i

                Para j=1 hasta Columnasroot
                        Si A(i, j) < threshold entonces
                                R1 = R1 + A(i, j)
                                cantPuntos1 = cantPuntos1 + 1
                        Si no
                                R2 = R2 + A(i, j)
                                cantPuntos2 = cantPuntos2 + 1
                }

        // Entre todos realizan la suma de todas las sumas parciales
        // El resultado queda en las mismas variables En R1, R2, cantPuntos1 y cantPuntos2
        // donde cada uno calculó son sumas locales
        MPI_AllReduce( R1, R1, .... , MPI_SUM, ....)
        MPI_AllReduce( R2, R2, .... , MPI_SUM, ....)
        MPI_AllReduce( cantPuntos1, cantPuntos1, .... , MPI_SUM, ....)
        MPI_AllReduce( cantPuntos2, cantPuntos2, .... , MPI_SUM, ....)

        // Cada uno realiza los cálculos finales en forma local
        m1Anterior = m1
        m2Anterior = m2
        m1 = R1 / cantPuntos1
        m2 = R2 / cantPuntos2
        threshold = (m1 + m2) / 2
        }
```

El código ejecutado por cada uno de los otros procesos es similar, con la diferencia que en lugar de enviar los datos como lo hace el raíz, recibe de aquél la información que le corresponde.

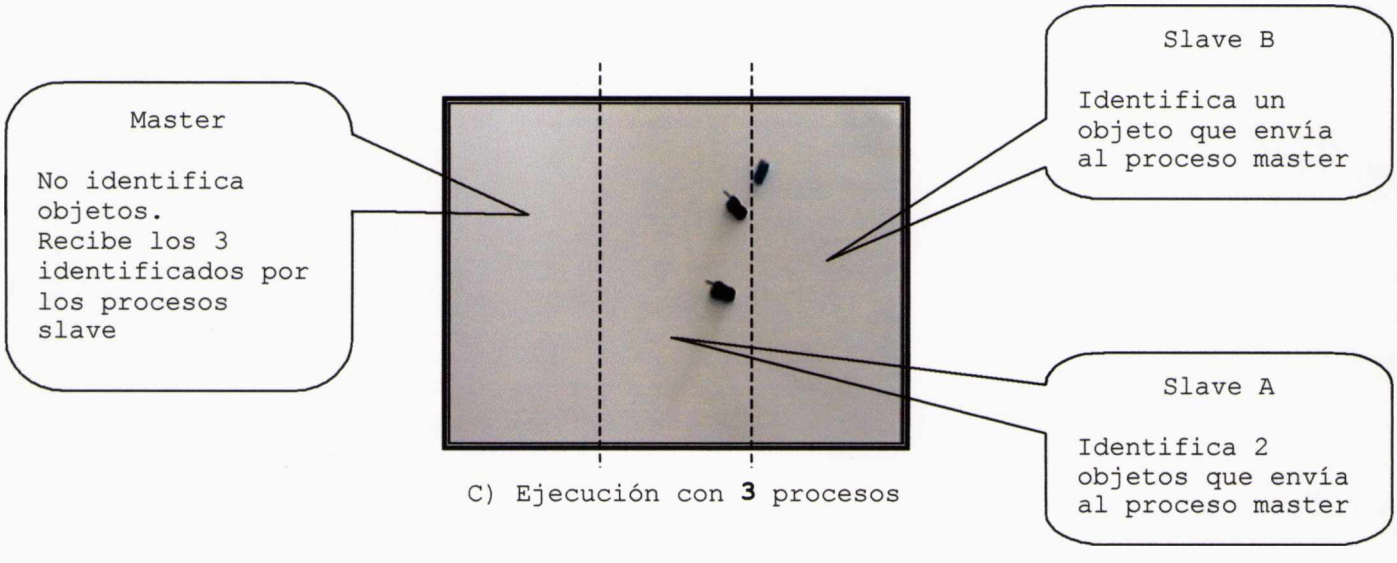
Notar que al terminar la operación de selección, todos los procesos tienen el resultado de la operación. Es necesario que así sea, ya que este valor es utilizado por cada uno durante la fase de reconocimiento de objetos.



A) Ejecución con **1** proceso

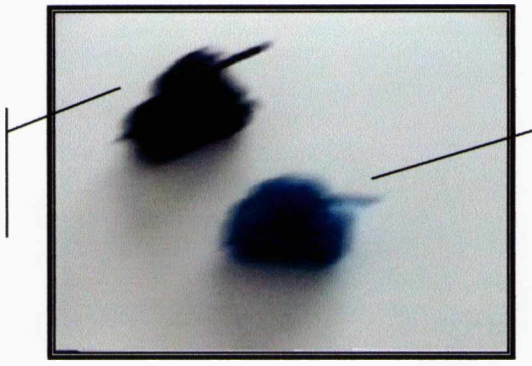


B) Ejecución con **2** procesos

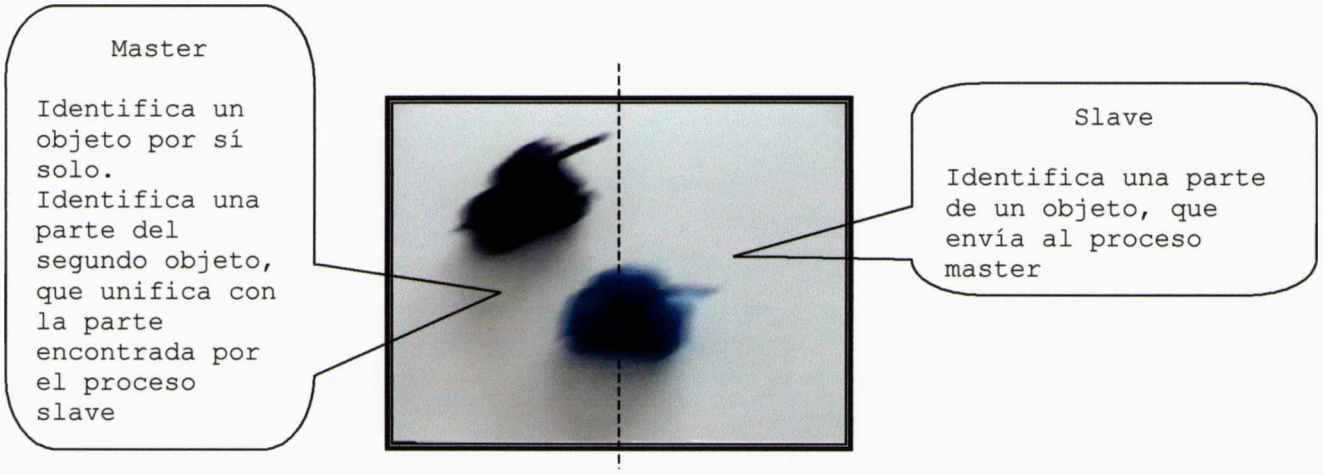


C) Ejecución con **3** procesos

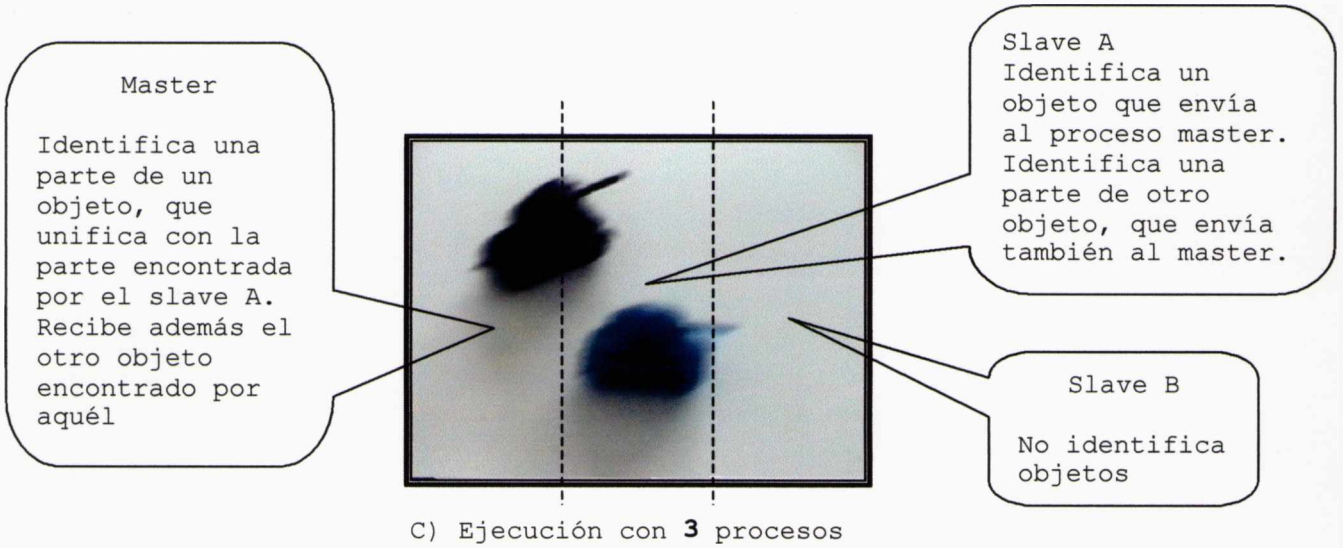
Ejemplo de reconocimiento de objetos con 1, 2 y 3 procesos



A) Ejecución con **1** proceso



B) Ejecución con **2** procesos



C) Ejecución con **3** procesos

Ejemplo de reconocimiento de objetos con 1, 2 y 3 procesos

Parte II - Seguimiento de Trayectorias

Ya se han identificado los objetos de las imágenes. Ahora sigue el armado de las trayectorias. Retómese la solución secuencial del problema. Siendo P_1, P_2, ..., P_k los conjuntos de objetos encontrados en una secuencia de *k* imágenes, para cada imagen P_i se tomaban los objetos y se los "enganchaba" en las trayectorias armadas para las imágenes anteriores siguiendo este algoritmo:

Para cada objeto existente en el cuadro k+1 crear una trayectoria fantasma

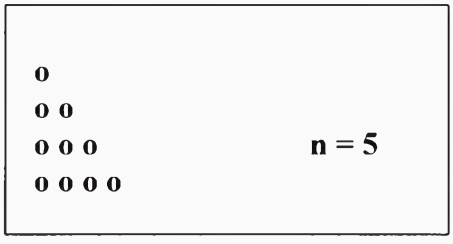
// Calcula la máxima desviación global
Para i=1 hasta m-1
    Para j=i+1 hasta m
        Si en el intercambio se cumplen las restricciones de d_{max}, calcular
            $G_{ij}^{k} = [\phi(P_i^{k-1}, P_i^{k}, P_i^{k+1}) + \phi(P_j^{k-1}, P_j^{k}, P_j^{k+1})] -$
                $[\phi(P_i^{k-1}, P_i^{k}, P_j^{k+1}) + \phi(P_j^{k-1}, P_j^{k}, P_i^{k+1})]$
    Tomar el par *ij* con máxima ganancia (G_{ij})

Si la ganancia G_{ij} es mayor que cero entonces
    Intercambiar los puntos del cuadro *k+1*, P_i^{k+1} con P_j^{k+1}.
Grabar el resultado de este paso en el archivo de trayectorias

donde **m** es el número máximo de objetos entre las 3 imágenes más recientes, más un número igual de trayectorias fantasmas. Se había visto que toda la información relacionada con las distintas trayectorias se iba guardando en un archivo de trayectorias a medida que se avanzaba en el procedimiento. En memoria principal siempre se tenía la información de los últimos tres cuadros por una cuestión de performance y de facilidad de procesamiento.
Repasando la solución dada, el procedimiento secuencial para realizar el seguimiento de trayectorias de objetos se puede resumir de la siguiente manera: Se trata de un ciclo repetitivo en el cual, a medida que se dispone de nuevos cuadros de una secuencia de imágenes, se actualizan las trayectorias armadas para los cuadros anteriores, con los objetos identificados en el nuevo cuadro. Es decir, sea la instancia donde hay que procesar el cuadro *i*. Se tienen N trayectorias incompletas, identificadas en los *i-1* cuadros anteriores. El siguiente paso es, entonces, "enganchar" en las trayectorias anteriores los *x* objetos identificados en el cuadro *i*. Para hacer esto, es necesario realizar varias operaciones de comparación para determinar cuál es la disposición de objetos que da un mejor conjunto de trayectorias.
Como resultado de estas operaciones probablemente haya que intercambiar el orden de algunos de los objetos hallados. Si es así, el ciclo se repite hasta que no haya más intercambios. Si *n* es el máximo entre la cantidad de trayectorias armadas, y el número de objetos encontrados en el nuevo cuadro, la cantidad de operaciones de comparación o verificación a realizar en cada ciclo es *(n-1) + (n-2) + ... + 1*. Es fácil comprobar que esta suma es igual a:

$$\sum_{i=1}^{n-1} i = [(n-1)^2 + n - 1]/2$$

Para eso observar el siguiente gráfico de muestra:



Cada punto representa una de las operaciones a realizar. Suponiendo que los puntos miden 1x1, la cantidad de puntos es igual a la superficie del triángulo por debajo de la diagonal que va desde el ángulo superior izquierdo hasta el ángulo inferior derecho, más la superficie de los puntos por arriba de esa diagonal. La primera es $[(n-1)^2] / 2$. Falta considerar los "medios puntos" que están por encima de la diagonal. La diagonal completa tiene (n-1) puntos, por lo tanto la medida de la mitad es (n-1) / 2. Juntando los dos términos se tiene la fórmula inicial.

Recuérdese que cada una de las operaciones a las que se esta haciendo mención es una función que consiste en lo siguiente: Se tienen dos ternas de puntos u objetos donde cada uno corresponde a una de tres imágenes consecutivas, y determina una posible trayectoria. La función verifica cuál combinación da una menor desviación total: Si las dos trayectorias dadas, o las que resultan de intercambiar el tercer punto de ambas ternas. Cuando el resultado es favorable para las ternas con el punto intercambiado, es cuando se debe realizar el intercambio mencionado.
Por supuesto la cantidad de intercambios de objetos y la cantidad de ciclos necesarios hasta que no haga falta más intercambios dependerán directamente del conjunto de datos de muestra.

Una paralelización inmediata consistiría en repartir equitativamente esas operaciones entre los distintos procesos. La forma de repartir sería como antes, es decir, en la medida de lo posible se repartiría entre todos la misma cantidad para cada uno. Si el número de operaciones no fuera múltiplo exacto del número de procesos, las operaciones sobrantes se repartirían una para cada uno de los primeros procesos. Por lo demás, todos realizarían las mismas funciones, con una copia local de los datos de las imágenes más recientes.

Los siguientes son algunos ejemplos de la cantidad de operaciones según el número de procesos:

N° de objetos	N° de operaciones	2 procesos	3 procesos	n procesos (*)
2	1	0.5	0.33	1/n
3	3	1.5	1	3/n
4	6	3	2	6/n
5	10	5	3.33	10/n
6	15	7.5	5	15/n

(*) OBS.: Nótese que la división de las tareas entre procesos se muestra como una división exacta con el fin de que el análisis sea más claro. Por supuesto en la implementación real, se reparte lo más equitativamente posible, lo cual no implica que a todos los procesos les corresponda exactamente la misma cantidad de tareas, ya que el número de tareas a repartir - en este caso son operaciones de comparación - no siempre es múltiplo exacto del número de procesos)

Para un número manejable de objetos – en particular para el caso del presente proyecto – el reparto de las operaciones resultaría en una cantidad minúscula de procesamiento a realizar por cada uno. Como desventaja, hay que tener en cuenta que además a esto se le suman los tiempos de envío de información entre procesos. Ante esta conclusión lo mejor es sencillamente descartar esta posibilidad y buscar una alternativa más adecuada.

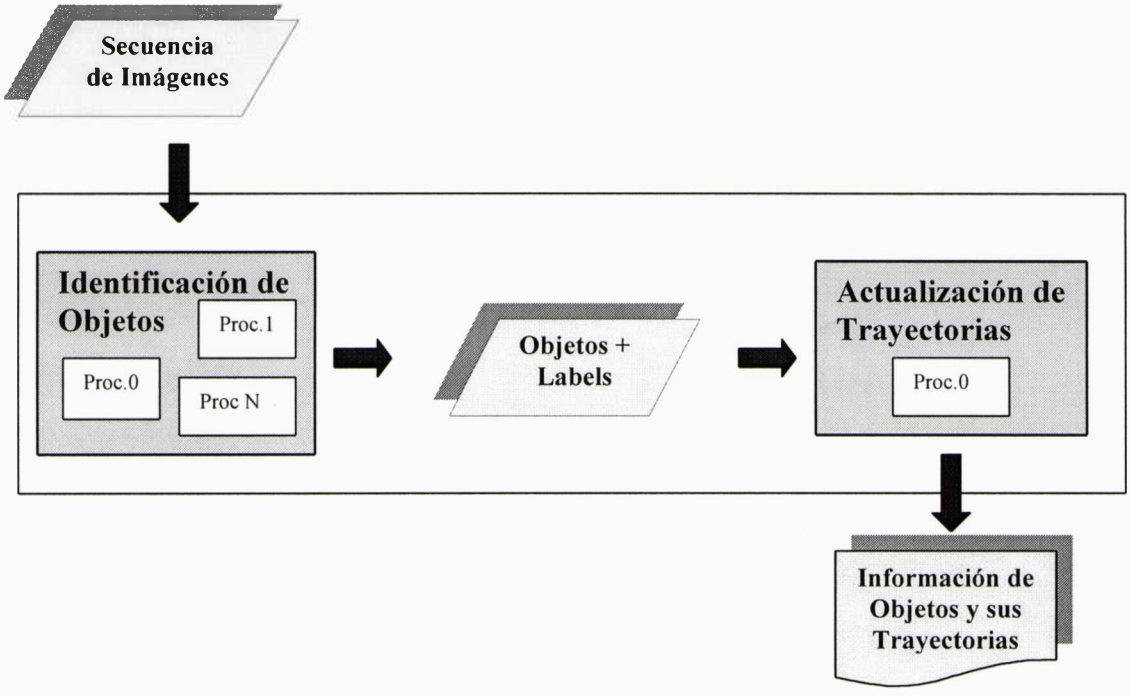
La idea que surge es mirar la solución desde un nivel de menos refinamiento. Esta posibilidad se explica a continuación.

El problema en su expresión general consiste, como se vio, en identificar objetos en una secuencia para luego seguir sus trayectorias. La actualización de las trayectorias puede hacerse siempre que se haya completado la identificación de objetos en al menos tres imágenes de la secuencia.
Al disponer de más de un proceso para resolver este problema, una solución que aproveche esta situación podría ser la siguiente: en lugar de intentar paralelizar el algoritmo interno del seguimiento de trayectorias, separar las dos tareas centrales entre los distintos procesos. Esto es, por un lado se tiene un conjunto de procesos que realizan la identificación de objetos, mientras que un proceso aparte se ocupa de actualizar las distintas trayectorias.

Con esta propuesta se logra entonces superponer la parte de actualización de trayectorias con la fase de identificación de objetos en la imagen siguiente. Si se viera que ese tiempo es muy pequeño y mantiene al proceso responsable ocioso por mucho tiempo, se podría decidir que esta parte del procesamiento la realizar uno de los procesos del primer grupo.

Parte III – Integrando la Información

Ya se vieron las distintas partes que componen la solución. En el siguiente esquema se representan todas las piezas juntas para dar una idea resumida de cómo funciona la solución planteada.



El marco identifica las tareas a realizar, las cuales son repetitivas. El proceso raíz – en este caso el *proceso 0* – ejecuta tanto la parte de identificación como la de actualización de trayectorias. El resto de los procesos únicamente ejecutan una y otra vez, según la cantidad de imágenes de la secuencia, la parte de identificación de objetos.

Si en el cuadro anterior correspondiente a la actualización de trayectorias se reemplaza "proc.0" por "proc.X", se tendría representado el caso en que se tiene a un proceso dedicado para realizar esta función. Este proceso repetiría este procedimiento cada vez que los otros procesos terminan de identificar los objetos en una nueva imagen.

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

EVALUACION DE LOS RESULTADOS

Introducción

El objetivo central del proyecto fue desarrollar algoritmos que permitan obtener resultados en tiempos aceptables considerando el origen del problema. En consecuencia para evaluar el cumplimiento de este objetivo se realizarán en esta sección distintas pruebas comparando la respuesta de las dos soluciones planteadas frente a una misma situación. Para simplificar esta etapa final se comenzará por comparar la performance de los algoritmos más elementales, para finalmente realizar una evaluación general.

Para poder medir los diferentes algoritmos en función de los tiempos de procesamiento se utilizaron las funciones de MPI ya mencionadas provistas para tal fin: MPI_time(), etc. Para poder medir ambas soluciones de la misma manera, la implementación secuencial fue compilada con la librería MPI y se incorporó al código la utilización de estas funciones.

En primer lugar se presentarán consideraciones generales que darán la base para realizar la evaluación sobre las soluciones planteadas. Después se detallará en particular la evaluación aplicada a cada parte de la solución y se presentarán los resultados obtenidos.

Generalidades

Speed-up y Eficiencia

Con el fin de evaluar la performance de las dos soluciones dadas, dos términos a tener en cuenta son: ***speed-up*** y ***eficiencia***, los cuales se explican más abajo.

La razón principal de tener múltiples procesadores es poder resolver un problema más rápido. Sería de esperar que cuantos más procesadores se utilicen más rápido se resolverá el problema. Sin embargo, en algún punto los tiempos de coordinación y comunicación entre procesos empiezan a dominar el problema con lo cual el resultado es el contrario al esperado. Idealmente se quisiera que la ineficiencia sea mínima. Si se toma el ideal como parámentro, se puede medir cuán lejos está la implementación de ese ideal.
En el mejor caso, el tiempo para ejecutar una solución decrece proporcionalmente al número de procesos. Esto es, si:

N = número de procesos
T_N = tiempo para usar el problema usando N procesos

a medida que N crece, T_N debería decrecer. En el caso ideal:

$$T_N = T_1 / N \quad \text{o} \quad N = T_1 / T_N$$

Esta última fórmula da un indicador de ***speed-up***, es decir que en el caso ideal, tener N procesos reduciría los tiempos N veces. En general:

$$T_1 / T_N \leq N$$

El cociente entre speed-up real en N procesos y el ideal, da una medida para la ***eficiencia***. Formalmente:

eficiencia = real / ideal = speed-up real en N procesos / N

La eficiencia se mide en un número ≤1, o como porcentaje: ≤100%. Intuitivamente, valores bajos de eficiencia representan desperdicio de potencia de procesamiento.

Tamaño del problema y tamaño de la máquina

En general para cada parte de la solución paralela el tiempo total de ejecución se puede dividir en dos componentes:
Por un lado está la parte del problema que fue paralelizada. Esta es la parte donde agregar procesadores realmente reduce los tiempos, incluso muy cerca del ideal. Sea T_p el tiempo ocupado para ejecutar esta parte del problema en la solución secuencial.
Va a haber asimismo otra parte del problema que es inherentemente secuencial o no paralelizable. Esta parte *no* va a ser más rápida agregando procesadores puesto que debe ser o bien ejecutada al menos una vez por cada proceso, o bien por un único proceso mientras el resto está ocioso esperando algún resultado del primero. Llamando T_S al tiempo que lleva ejecutar esta parte del problema, se tiene:

$$T_1 = T_S + T_p$$

T_S es un límite inferior inamovible para los tiempos de procesamiento; es decir, $T_N \langle T_S$ independientemente de N. Se puede reducir el tiempo T_p agregando más procesos, pero no se puede reducir T_S. Por lo tanto se puede pretender en el mejor de los casos tener la siguiente igualdad:

$$T_N = T_S + (T_p / N)$$

El problema clave es que la fracción del tiempo total de ejecución correspondiente a la parte no paralelizable, T_S / T_1, impone un límite en la reducción de los tiempos independientemente del número de procesos. Entonces cuanto más grande sea esta fracción, menos eficiente será la solución paralela que se proponga. Es importante tener en cuenta que este factor no es una constante, sino que varía con el tamaño del problema. A medida que el tamaño del problema cambia, el factor T_S / T_1 cambia también; típicamente T_p crece mucho más rápido que T_S.

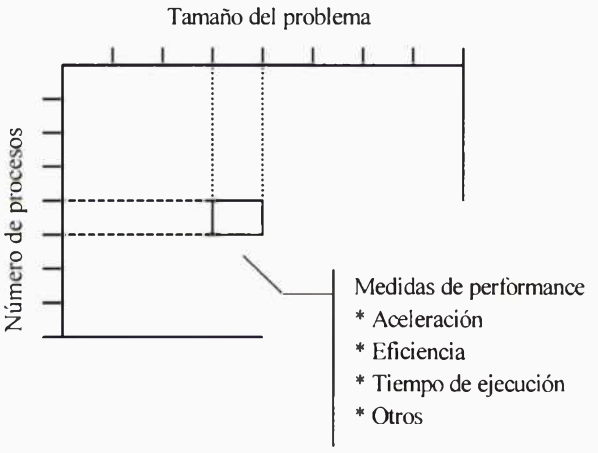
Como consecuencia de lo planteado, queda claro que una completa evaluación debería extenderse al menos sobre dos parámetros

(1) El tamaño del problema
(2) El número de procesadores

que son los principales factores que influyen en los resultados finales. A esto hay que sumar el hecho de que la performance puede ser seriamente afectada tanto por operaciones de I/O, como por la cantidad de comunicaciones entre procesos.

Los conceptos vistos implican que una completa evaluación debe balancear factores de eficiencia, speed-up, costos de hardware y el tamaño del problema a ser resuelto. Se deben

considerar distintos casos tanto sobre el rango de tamaños de problema sobre el cual se ejecutará la aplicación, como sobre el rango de tamaños de máquina o número de procesos utilizados. Esta idea se grafica a continuación:



La Evaluación

Para poder determinar las características que debían tener las secuencias de imágenes de modo que la evaluación contemplara todas las situaciones posibles, se revisaron primeramente los algoritmos correspondientes a cada una de las partes del sistema. A partir de esta revisión se definieron ***13*** casos de prueba y se generaron las secuencias de imágenes necesarias (*). Seguidamente se evaluaron los algoritmos sobre cada uno de estos casos de prueba variando la cantidad de procesos y la cantidad de máquinas involucrados.

(*) El APENDICE E: Pruebas realizadas y resultados de la evaluación describe cada una de las pruebas realizadas incluyendo un detalle de los resultados obtenidos.

Disposición de los procesos en las máquinas

Todo el conjunto de pruebas fue realizado disponiendo un proceso por máquina. Una alternativa hubiera sido distribuir los procesos de modo que en una misma máquina hubiera más de uno de ellos, pero realizando varias pruebas se pudo ver que esta opción desvirtuaría los resultados. Esto se puede ver con el siguiente ejemplo, que muestra los tiempos obtenidos para la ejecución del sistema sobre los datos de la prueba número uno explicada en el apéndice:

Procesos	720 x 540	
	Tiempo total	Tiempo x imagen
1	115.442883	*8.880222*
2	102.552633	*7.888664*
3 - CASO A	102.552633	*7.888664*
3 - CASO B	525.658837	*40.435295*
3 - CASO C	115.988133	*8.922164*

donde:

- Caso A: corresponde a tres procesos ejecutando en tres máquinas diferentes.
- Caso B: corresponde al master y un slave ejecutando en una máquina, más otro slave ejecutando en una máquina diferente.
- Caso C: corresponde al master ejecutando solo en una máquina, más dos slaves ejecutando en una segunda máquina.

Como se observa, el *caso B* es el peor en performance y resulta incomparable con cualquiera de las restantes situaciones. El *caso C* por otro lado muestra resultados similares a los de la prueba con un único proceso, con la diferencia que en este caso, al estar participando dos máquinas, se está desperdiciando la mitad de su poder de procesamiento.

Es debido a este tipo de resultados que todas las pruebas se ejecutaron con una distribución pareja de un proceso por máquina.

A continuación se explica para cada parte del sistema, con el razonamiento que llevó a decidir cuáles eran los factores influyentes en la performance de cada algoritmo. Se dibujan tablas que resumen los resultados obtenidos a partir de las diferentes pruebas. Las observaciones y conclusiones que complementan esta evaluación tienen que ver con el comportamiento general que se revele a partir de estos resultados.

Realce de una Imagen

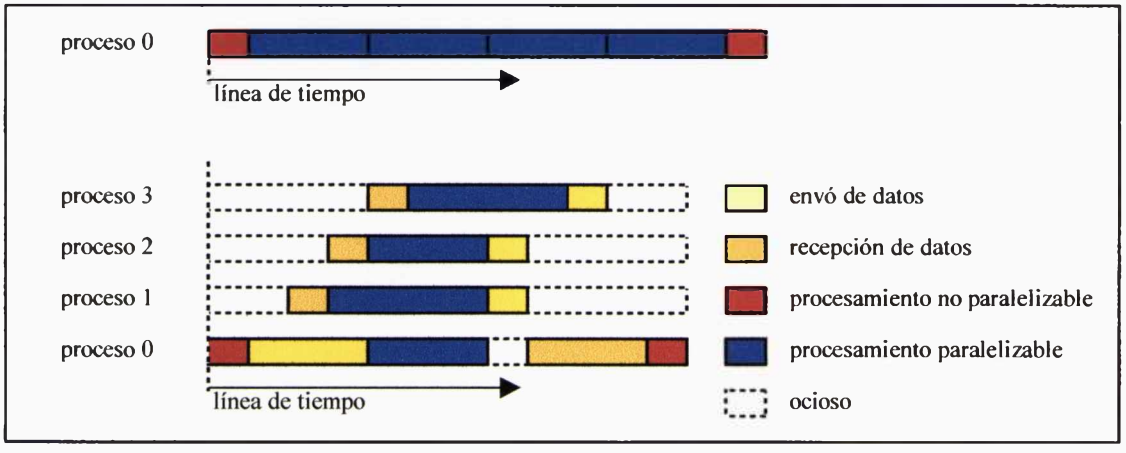
Retómense las soluciones vistas. En esta implementación el proceso de realce consiste en recorrer toda la imagen original y por cada punto calcular el valor medio sobre una ventana de cinco puntos. La imagen resultante se construye a partir de estos valores medios.
Para paralelizar la función lo que se hizo fue repartir la imagen entre los distintos procesos, asignando a cada uno de ellos la tarea de generar una parte de las columnas de la nueva imagen.

Siguiendo la terminología anterior, se trata de identificar acá la parte del problema que insume los tiempos T_S y T_p, para poder analizar cómo influyen en la solución propuesta.

1) *Tiempo* T_S. El tiempo fijo, no paralelizable, incluye la obtención de la imagen, en este caso apertura y cierre de archivos de imagen, y el reconocimiento del formato de la imagen, por ejemplo, averiguar sus dimensiones.

2) *Tiempo* T_p. El tiempo correspondiente a la parte paralelizable es el que toma calcular el valor para cada punto de la nueva imagen. En el caso de más de un proceso, este tiempo no es puramente tiempo de procesamiento de la imagen, sino que lleva consigo los tiempos de envío y recepción de estos datos entre los distintos procesos.

El proceso se puede representar en un gráfico temporal de la siguiente manera:



En la parte superior del gráfico se muestra el caso secuencial, mientras que abajo se representan los tiempos para un ejemplo de ejecución de la solución paralela con cuatro procesos.

* Nota: En el ejemplo los tiempos de procesamiento de los distintos procesos se dibujaron diferentes. Esto no quiere decir que entre ellos tengan distinta cantidad de datos a procesar sino simplemente se quiso mostrar que según la máquina donde cada uno ejecute, el tiempo de procesamiento puede variar.

El tamaño del problema en este caso es directamente proporcional al tamaño de la imagen que se está procesando. Cuantas más filas y columnas tenga la imagen, tantos puntos o pixels

más hay que calcular para la imagen resultante. Si se tiene en cuenta que cada proceso recibe además de las columnas de datos a procesar, las dos columnas adyacentes a éstas, puede deducirse que cuantos más procesos haya, más tráfico de datos se generará. Pero al mismo tiempo, el agregado de procesos supone una disminución en tiempo de procesamiento por proceso. Por lo tanto es necesario buscar un equilibrio entre estos dos puntos.

Para medir la performance de las soluciones propuestas para el realce de imágenes se generaron casos de prueba con:
- diferentes tamaños de imagen (tamaño del problema)
- distintas cantidades de procesos

El siguiente cuadro sintetiza los resultados obtenidos en la evaluación. Los números corresponden al tiempo promedio por imagen sobre el total de las pruebas realizadas. La unidad de medida son segundos.

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.673101*	*Tiempo:*	*3.235257*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.692028*	*Tiempo:*	*2.630616*
	Speed-up:	0.972649	Speed-up:	1.229848
	Eficiencia:	48.63%	Eficiencia:	61.49%
3	*Tiempo:*	*0.838807*	*Tiempo:*	*2.792070*
	Speed-up:	0.802450	Speed-up:	1.158731
	Eficiencia:	26.75%	Eficiencia:	38.62%

Si se observan los resultados del apéndice correspondiente, se puede ver que este cuadro muestra prácticamente el reflejo de cada prueba individual. Es decir, independientemente del tipo de imágenes con que se trate, los tiempos que consume la tarea del realce son similares. Lo primero que se puede ver es que una secuencia de imágenes de tamaño chico no es la mejor candidata para ser realzada en paralelo. En este caso los tiempos que le toma a un solo proceso realizar la operación son mejores que los que resultan de la parelelización.
Por otro lado, para un tamaño de imagen bastante más grande, se puede ver que el speed-up es de un 22% sobre el tiempo base, al incorporar un segundo proceso; no es un resultado óptimo, pero resulta positivo. No ocurre lo mismo agregando un tercer proceso, con el cual los tiempos de procesamiento vuelven a subir.

Reconocimiento de Objetos

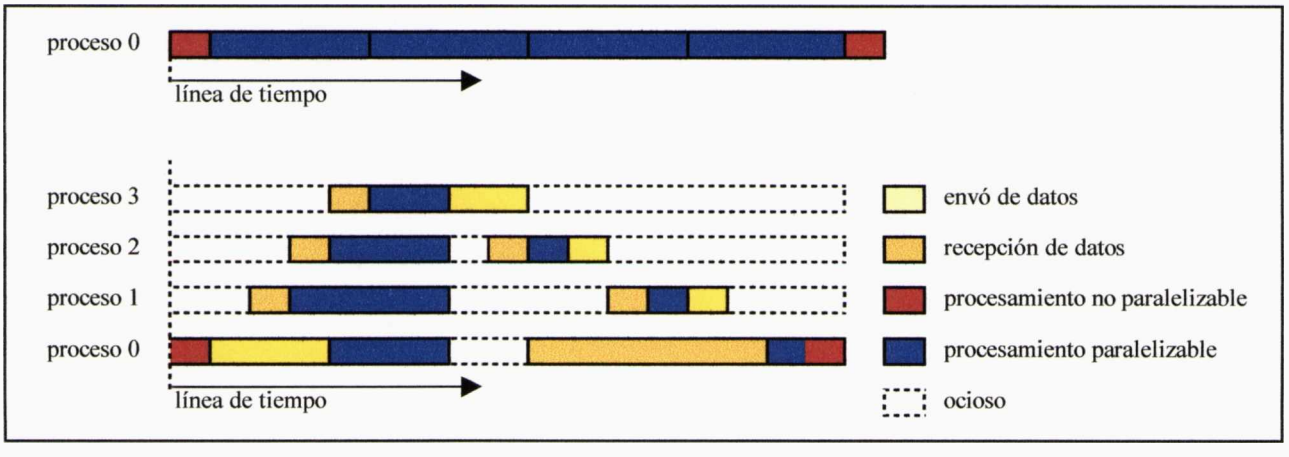
En la implementación presente tal como se vio en los párrafos anteriores, el proceso de reconocimiento de objetos se simplifica a la identificación de zonas claramente distinguibles de un fondo contrastante. Para llevar a cabo este proceso se realiza una segmentación de la imagen separando la zona ocupada por objetos de la zona del fondo. Al mismo tiempo que se realiza esta segmentación, los puntos de la imagen se van etiquetando de manera de poder individualizar en la zona de objetos a los distintos componentes de la escena.
Similarmente a lo realizado para el proceso de realce de una imagen, en esta caso la paralelización también empieza con una división de la imagen por columnas entre los distintos procesos, tras lo cual cada uno analiza esa parte buscando los objetos que allí se pudieran encontrar. Después de haber realizado la tarea individual, entre todos los procesos realizan una tarea conjunta para ensamblar las distintas partes de los objetos que pudieron haber quedado "partidos" al dividir la imagen. Por último, todos los procesos envían la información obtenida a un proceso principal.

Los siguientes son los tiempos T_S y T_p ya descriptos, que componen el tiempo total de procesamiento:

3) *Tiempo* T_S. El tiempo fijo - no paralelizable - incluye, igual que en el caso anterior, la obtención de la imagen, y el reconocimiento del formato de la imagen.

4) *Tiempo* T_p. El tiempo correspondiente a la parte paralelizable es el que toma identificar los objetos en la imagen, pero hay que contemplar en el caso de la solución paralela que este tiempo lleva consigo el envío de datos entre procesos más el eventual "ensamblado" de objetos que posiblemente hayan quedado divididos en el reparto de la imagen.

El proceso de reconocimiento de objetos se puede representar en un gráfico temporal de la siguiente manera:



Igual que antes, en la parte superior del gráfico se muestra el caso secuencial, mientras que abajo se representan los tiempos para un ejemplo de ejecución de la solución paralela con cuatro procesos.

El tamaño del problema en este caso depende de dos factores principales:

1) el tamaño de la imagen que se está procesando. Cuantas más filas y columnas tenga la imagen, tantos puntos o pixels más hay que inspeccionar para encontrar la información de los objetos.
2) El "juego de datos" particular que se esté analizando. Concretamente, la cantidad, el tamaño y la disposición de los objetos en cada imagen influyen directamente en el tiempo de procesamiento, y en la cantidad de información que tendrá que "viajar" entre los procesos.

Por ejemplo, un problema grande estará caracterizado por imágenes de alta resolución, o bien con muchos objetos, o bien con objetos cuya forma sea alargada de manera que al dividir la imagen los mismos queden "partidos" entre los distintos procesos.

Con respecto a la cantidad de procesos, al igual que en el caso anterior, al repartirse dos columnas más por proceso, cuantos más procesos haya, más tráfico de datos se generará. Pero por otro lado cada uno tiene menos datos a inspeccionar cuando el número de procesos crece. La desventaja es que al estar más partida la imagen, existe la posibilidad de partir más los objetos, lo que agrega tiempo de "ensamblado" de objetos al final.

Para medir la performance de las soluciones propuestas para el reconocimiento de objetos en imágenes se generaron casos de prueba con:

- imágenes que generen distintos tamaños de problema tal como se explicó arriba.
- distintas cantidades de procesos

Las distintas pruebas fueron agrupadas según las características mencionadas, de modo que la evaluación resulte más interesante. Así se armaron tablas resumen para los dos grupos siguientes:

I. Pruebas agrupadas por cantidad de objetos
II. Pruebas agrupadas por tamaño de los objetos

Se debe notar que no se consideró como criterio de agrupamiento la disposición de los objetos – es decir, su ubicación dentro de la imagen – puesto que al tratarse de secuencias donde los objetos se mueven, dentro de una misma prueba esta característica no es fija; los objetos están en un momento en un sector de la imagen que puede ser muy distinto del lugar donde se encuentran al final de la secuencia.

I. Pruebas agrupadas por cantidad de objetos

a) Un objeto *(pruebas 1, 7 y 9)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.811783*	*Tiempo:*	*8.192109*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.387997*	*Tiempo:*	*6.433421*
	Speed-up:	0.584859	Speed-up:	1.273367
	Eficiencia:	29.24%	Eficiencia:	63.67%
3	*Tiempo:*	*1.320667*	*Tiempo:*	*4.324453*
	Speed-up:	0.614676	Speed-up:	1.894369
	Eficiencia:	20.49%	Eficiencia:	63.15%

b) Dos objetos *(pruebas 8, 10, 11 y 12)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*2.619058*	*Tiempo:*	*23.438881*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.456582*	*Tiempo:*	*16.222862*
	Speed-up:	1.066139	Speed-up:	1.444806
	Eficiencia:	53.31%	Eficiencia:	72.24%
3	*Tiempo:*	*1.951861*	*Tiempo:*	*10.371249*
	Speed-up:	1.341826	Speed-up:	2.259986
	Eficiencia:	44.73%	Eficiencia:	75.33%

c) Tres o más objetos *(pruebas 2, 3, 4, 5, 6 y 13)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*1.062799*	*Tiempo:*	*9.806780*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.446556*	*Tiempo:*	*7.250074*
	Speed-up:	0.734710	Speed-up:	1.352646
	Eficiencia:	36.74%	Eficiencia:	67.63%
3	*Tiempo:*	*1.414977*	*Tiempo:*	*5.221229*
	Speed-up:	0.751107	Speed-up:	1.878251
	Eficiencia:	25.04%	Eficiencia:	62.61%

II. Pruebas agrupadas por tamaño de los objetos

a) Objetos pequeños *(pruebas 1 a 6)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.596855*	*Tiempo:*	*6.257396*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.158318*	*Tiempo:*	*5.035829*
	Speed-up:	0.515278	Speed-up:	1.242575
	Eficiencia:	25.76%	Eficiencia:	62.13%
3	*Tiempo:*	*1.243083*	*Tiempo:*	*3.732283*
	Speed-up:	0.480141	Speed-up:	1.676560
	Eficiencia:	16.00%	Eficiencia:	55.89%

b) Objetos medianos *(pruebas 7 y 8)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.888730*	*Tiempo:*	*8.578558*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.324903*	*Tiempo:*	*6.293947*
	Speed-up:	0.670788	Speed-up:	1.362986
	Eficiencia:	33.54%	Eficiencia:	68.15%
3	*Tiempo:*	*1.337735*	*Tiempo:*	*4.585571*
	Speed-up:	0.664354	Speed-up:	1.870772
	Eficiencia:	22.15%	Eficiencia:	62.36%

c) Objetos grandes *(pruebas 9 a 13)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*2.785956*	*Tiempo:*	*24.494208*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.613988*	*Tiempo:*	*16.977858*
	Speed-up:	1.065788	Speed-up:	1.442715
	Eficiencia:	53.29%	Eficiencia:	72.14%
3	*Tiempo:*	*2.025067*	*Tiempo:*	*10.844177*
	Speed-up:	1.375736	Speed-up:	2.258743
	Eficiencia:	45.86%	Eficiencia:	75.29%

Las pruebas con respecto al algoritmo de reconocimiento de objetos revelan mejores resultados que en el caso anterior, aunque siempre en el caso de las imágenes de mayor tamaño.

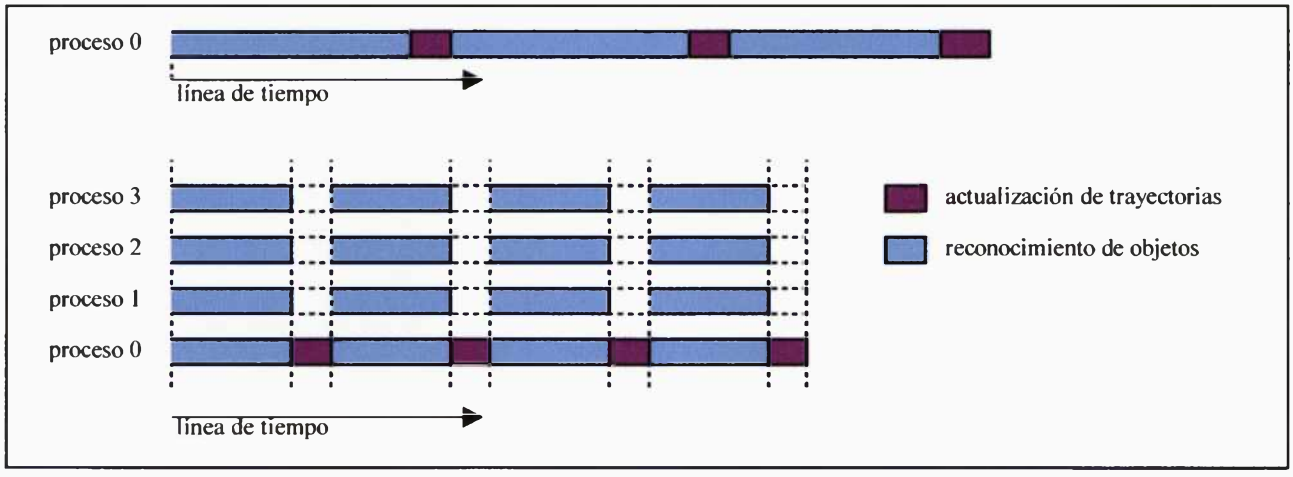
En el caso de las secuencias de imágenes chicas, en la mayoría de los casos los tiempos aumentan junto con la cantidad de procesos. El único caso donde esto no ocurre es cuando las imágenes contienen objetos grandes, pero el beneficio es muy poco y la eficiencia de las máquinas baja tanto que el resultado final resulta muy pobre.

Para el caso de las imágenes más grandes en cambio, la incorporación de procesos mejora los tiempos, cosa que se destaca principalmente en los casos en que las imágenes contienen objetos grandes. Además se puede observar que en general el speed-up que se logra al incorporar un tercer proceso es mejor aún que el que se obtiene al pasar de uno a dos procesos.

Seguimiento de Trayectorias

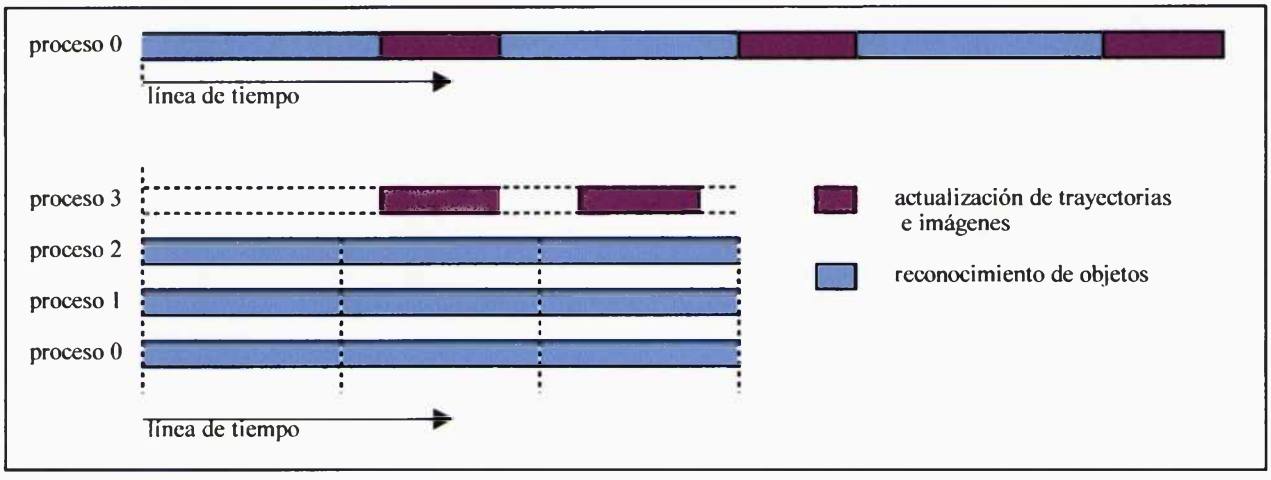
El proceso de seguimiento de trayectorias de objetos en una secuencia de imágenes consiste en tomar el resultado del reconocimiento de objetos en las distintas imágenes, e identificar la correspondencia de los objetos entre cada nueva imagen y las anteriores. La solución paralela que se dio a este procedimiento fue asignar varios procesos a la tarea de reconocimiento de los objetos y cada vez que se completa el reconocimiento para una imagen, un proceso del mismo conjunto se ocupa de realizar la correspondencia de objetos con el fin de actualizar las trayectorias.

Representada en forma temporal, la situación sería la siguiente:



La situación se asemeja bastante a la vista para el caso de reconocimiento de objetos. La diferencia la marca la parte de procesamiento que debe realizar el proceso maestro o raíz luego de analizada cada imagen. Por esto para responder acerca de la dimensión del problema, son válidos los mismos conceptos planteados para el caso del reconocimiento de objetos: cantidad de objetos, disposición, etc. Además, dado que la cantidad de operaciones a realizar - en cuanto a la actualización de las trayectorias se refiere - depende de las características de los movimientos de los objetos, la velocidad y la disparidad con que éstos se muevan, influirán también en el tiempo de ejecución total de las soluciones tanto secuencia como paralela.

La tarea puntual de actualización de trayectorias, como ya se evaluó, es una fracción muy chica de todas las tareas involucradas en el problema. Por esto, no tendría sentido dedicar un proceso exclusivamente para que la realice. Sí podría hacerse esto en el caso que se planteó como mejora: Generar imágenes que muestren gráficamente la trayectoria que van dibujando los objetos en movimiento. En ese caso un proceso aparte actualizaría las trayectorias y también generaría estas imágenes, dando un gráfico como el de la siguiente página.



Para medir la performance del proceso de seguimiento de trayectorias se generaron secuencias de imágenes que varíen la dimensión del problema:

- secuencias con dos o tres objetos
- secuencias con más de tres objetos
- secuencias con objetos con movimientos lentos y parejos
- secuencias con objetos con movimientos rápidos y dispares

De las misma manera que en el caso del reconocimiento de objetos, las pruebas fueron agrupadas según las características mencionadas, en este caso de la siguiente manera:

I. Pruebas agrupadas por la velocidad y coherencia de los movimientos
II. Pruebas agrupadas por cantidad de objetos

Es importante tener en cuenta que las pruebas del algoritmo de seguimiento de trayectorias se refieren a la prueba integral que va desde el realce de cada cuadro individual hasta llegar al resultado final. No se trata de la prueba del algoritmo exclusivo del seguimiento.

I. Pruebas agrupadas por la velocidad y coherencia de los movimientos

a) Movimiento lento y parejo *(pruebas 7, 8, 9 y 10)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*3.504587*	*Tiempo:*	*19.058128*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.688455*	*Tiempo:*	*12.973783*
	Speed-up:	1.303569	Speed-up:	1.468972
	Eficiencia:	65.18%	Eficiencia:	73.45%
3	*Tiempo:*	*2.365918*	*Tiempo:*	*9.557698*
	Speed-up:	1.481280	Speed-up:	1.994008
	Eficiencia:	49.38%	Eficiencia:	66.47%

b) Velocidad moderada y pareja *(pruebas 4, 5 y 13)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*3.048173*	*Tiempo:*	*16.693136*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.496390*	*Tiempo:*	*12.469008*
	Speed-up:	1.221032	Speed-up:	1.338770
	Eficiencia:	61.05%	Eficiencia:	66.94%
3	*Tiempo:*	*2.391248*	*Tiempo:*	*9.356509*
	Speed-up:	1.274721	Speed-up:	1.784120
	Eficiencia:	42.49%	Eficiencia:	59.47%

c) Velocidad alta y movimiento parejo *(pruebas 1, 2, 6 y 11)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*2.692442*	*Tiempo:*	*14.215440*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.346723*	*Tiempo:*	*11.344055*
	Speed-up:	1.147320	Speed-up:	1.253118
	Eficiencia:	57.37%	Eficiencia:	62.66%
3	*Tiempo:*	*2.303768*	*Tiempo:*	*8.480294*
	Speed-up:	1.168712	Speed-up:	1.676291
	Eficiencia:	38.96%	Eficiencia:	55.88%

d) Velocidad y coherencia dispares *(pruebas 3 y 12)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*3.695427*	*Tiempo:*	*18.938554*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.824675*	*Tiempo:*	*14.414437*
	Speed-up:	1.308266	Speed-up:	1.313860
	Eficiencia:	65.41%	Eficiencia:	65.69%
3	*Tiempo:*	*2.618723*	*Tiempo:*	*11.027871*
	Speed-up:	1.411156	Speed-up:	1.717335
	Eficiencia:	47.04%	Eficiencia:	57.24%

II. Pruebas agrupadas por cantidad de objetos

a) Uno o dos objetos *(pruebas 1, 7, 8, 9, 10, 11 y 12)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*3.815133*	*Tiempo:*	*20.444036*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.830447*	*Tiempo:*	*14.676605*
	Speed-up:	1.347891	Speed-up:	1.392968
	Eficiencia:	67.39%	Eficiencia:	69.65%
3	*Tiempo:*	*2.499670*	*Tiempo:*	*10.636587*
	Speed-up:	1.526255	Speed-up:	1.922049
	Eficiencia:	50.88%	Eficiencia:	64.07%

b) Más de dos objetos *(pruebas 2, 3, 4, 5, 6 y 13)*

Procesos	320x240		720 x 540	
1	*Tiempo:*	*2.436260*	*Tiempo:*	*12.990423*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.244351*	*Tiempo:*	*10.128502*
	Speed-up:	1.085508	Speed-up:	1.282561
	Eficiencia:	54.28%	Eficiencia:	64.13%
3	*Tiempo:*	*2.265374*	*Tiempo:*	*7.970188*
	Speed-up:	1.075434	Speed-up:	1.629877
	Eficiencia:	35.85%	Eficiencia:	54.33%

Tal como se mencionó anteriormente, estas pruebas abarcan todo el ciclo del procesamiento de las imágenes. Como la porción exclusiva del mantenimiento de las trayectorias es una parte muy pequeña del algoritmo, y además es tarea de un único proceso, es de esperar que el resultado no se revierta respecto de las pruebas anteriores sobre el realce y el reconocimiento de objetos sobre las imágenes. Esto fue de hecho lo que ocurrió y se puede observar en los cuadros anteriores.

Resumiendo se tienen los siguientes resultados:

- ✓ Si se tienen imágenes chicas – en particular el tamaño analizado de 320 x 240 pixels – no resulta adecuado paralelizar los algoritmos, ya que en proporción resulta más el tiempo de pasaje de información entre procesos que el que cada uno ocupa después en realizar las distintas tareas. El trabajar con más de un proceso solamente bajó los tiempos en aquellos casos en que los objetos de las imágenes eran grandes, pero el speed-up resultó tan bajo que ni siquiera en estos casos merece tener en cuenta estos algoritmos paralelos para estas imágenes.

- ✓ Para el caso de tener imágenes más grandes – en particular de 720 x 540 pixels como en el caso de las pruebas realizadas – los resultados fueron aceptables no tanto en el

algoritmo inicial de realce de imágenes, sino en particular en el de reconocimiento de objetos y por supuesto en la prueba conjunta de todo el sistema. En especial, los resultados fueron mejores en los casos en que las imágenes contenían objetos de tamaño grande en relación al tamaño del cuadro. Por último, al momento de evaluar los algoritmos según los tipos de movimiento de los objetos, se observa en los casos donde los mismos son más lentos y parejos, los resultados son mejores aunque no por mucha diferencia con respecto a los demás.

Conclusiones Finales

Luego de haber completado la implementación y las evaluaciones de los algoritmos descriptos, el resultado obtenido cumplió con las expectativas planteadas en la introducción de la propuesta:

- Se desarrolló por completo una solución para identificar objetos en una secuencia de imágenes e identificar las trayectorias de sus movimientos.

- Se identificaron los casos en que la solución resulta una buena idea, y aquellos en los que es preferible adoptar la solución tradicional.

Por otro lado, a lo largo del proyecto se explicaron mejoras o alternativas para la solución planteada. Las mismas se resumen aquí junto con algunas sugerencias adicionales para el caso en que eventualmente las mismas se llevaran a cabo:

1) **Con respecto al proyecto**

- Generar imágenes adicionales que reflejen el movimiento de los objetos de la escena.

- Identificar objetos basados en características que los distingan unos de otros, es decir, buscar modelos de objetos. Como se vio, en la presente propuesta los objetos eran indivisibles. Esta mejora se complementa con una modificación en el algoritmo de seguimiento de trayectorias, que aparee los objetos no solo por su posición en los cuadros consecutivos sino también por las características propias que los individualizan.

- En el caso en que se tenga que seguir solamente a un objeto, modificar el algoritmo de reconocimiento de manera de inspeccionar en lugar de toda la imagen, solamente un sector dependiendo de dónde se haya encontrado el objeto en los cuadros anteriores. Esto mejora la solución, sin desperdiciar tiempo de procesamiento.

- Modificar o mejorar cualquiera de los algoritmos planteados para responder mejor según las distintas secuencias de imágenes que se quieran evaluar. Como se mencionó anteriormente, el proyecto está armado de manera tal que resulta simple cambiar cualquiera de sus partes por otra que responda mejor a los requerimientos. Por ejemplo se puede querer cambiar el algoritmo de reconocimiento de imágenes, o quizás se puede querer trabajar con otro formato de imágenes diferente al que se utilizó aquí.

2) **Con respecto a la performance**

- Evaluar cada cuánto tiempo es necesario realizar la selección del threshold para la segmentación explicada oportunamente. Dada la natural suavidad en los movimientos de las imágenes, no es necesario recalcular este valor para cada cuadro ya que no se obtendrían cambios significativos de una imagen a la siguiente. Sin embargo, la frecuencia necesaria puede depender de la velocidad de los movimientos de los objetos involucrados.

- Implementar una alternativa que maneje toda la información auxiliar como son los archivos de labels y objetos en memoria en lugar de archivo. Esta opción sería óptima en tiempos, pero requeriría disponer de las máquinas provistas de memoria suficiente que permita ejecutar la solución sin inconvenientes.

- Evaluar la posibilidad de saltear cuadros en el caso en que se conozca de antemano que los objetos en la escena se mueven muy lentamente. Como se mencionó en el inciso anterior y como se pudo ver en las pruebas realizadas, las imágenes consecutivas en los casos donde los objetos se mueven muy lentamente, no manifiestan diferencias significativas. Saltear los cuadros permitiría en estos casos llegar a la misma información pero con mejores tiempos.

- Para el caso que se conozca que los objetos son de tamaño considerable, implementar una variante que no inspeccione cada pixel sino que tome uno de cada dos o más, de modo de reducir el procesamiento. Se espera que los resultados no cambien, pero probablemente los tiempos sí lo hagan.

- Variar la distribución de las tareas entre los distintos procesos y evaluar los resultados. Por ejemplo, permitir que el realce lo realicen dos procesos, mientras que la tarea de reconocimiento de objetos la realicen entre tres. Esta distribución tiene que ver con aprovechar las conclusiones obtenidas de las pruebas del presente proyecto, o pruebas nuevas que pudieran realizarse. Por otro lado si se dispusiera de más máquinas para el procesamiento, otra posibilidad es distribuir la tarea de modo que un grupo realice el realce, y otro grupo de procesos en las otras máquinas, realice la parte de reconocimiento de objetos.

APENDICES

APENDICE A: REFERENCIA DEL LENGUAJE MPI

I) **Tipos de datos**. Los tipos de datos básicos en MPI se corresponden con los tipos de datos básicos del lenguaje host, en este caso "C". A continuación se listan los posibles valores para un tipo de dato, y el correspondiente tipo de dato en "C":

Tipo de dato MPI	tipo de dato "C"
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	insigned int
MPI_UNSIGNED_LONG	insigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

II) **Funciones**. A continuación se enumeran - en orden alfabético - las funciones de MPI utilizadas en la implementación, con su sintaxis y una breve descripción de su significado.

- `int MPI_ABORT(MPI_Comm comm, int errcode)`

Aborta todas las tareas en el grupo de procesos pertenecientes al comunicador `comm`. En `errcode` retorna un código de error.

- `int MPI_Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)`

Combina los elementos del buffer de entrada `sendbuf` de cada proceso del grupo usando la operación `op`, y retorna el valor resultante en el buffer de salida `recvbuf` de cada uno de los procesos del grupo. Las siguientes son las operaciones predefinidas que se pueden utilizar como parámetro `op`:

Nombre	Significado
MPI_MAX	máximo
MPI_MIN	mínimo
MPI_SUM	suma
MPI_PROD	producto
MPI_LAND	and lógico

MPI_BAND	and bit a bit
MPI_LOR	or lógico
MPI_BOR	or bit a bit
MPI_LXOR	xor lógico
MPI_BXOR	xor bit a bit
MPI_MAXLOC	máximo valor y ubicación
MPI_MINLOC	mínimo valor y ubicación

- `int MPI_Barrier(MPI_Comm comm)`

Sincronización de barrera. Bloquea el llamante hasta que todos los miembros del grupo han llamado a la función.

- `int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MI_Comm comm)`

Envía un mensaje desde el proceso con valor de rank `root` a todos los procesos del grupo.

- `int MPI_Comm_rank(MPI_Comm comm, int *rank)`

Indica el valor de `rank` del proceso que la invoca, en el rango de 0 a `size-1`, donde `size` es el valor retornado por `MPI_Comm_size`.

- `int MPI_Comm_size(MPI_Comm comm, int *rank)`

Indica el número de procesos involucrados en el comunicador `comm`.

- `int MPI_Finalize(void)`

Indica que se termina de ejecutar las operaciones de MPI.

- `int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)`

Toma como entrada la estructura que contiene el resultado de una operación `MPI_Recv` y retorna en `count` el número efectivo de datos recibidos por esa operación.

- `int MPI_Get_processor_name(char *name, int *resultlen)`

Retorna en `name` el nombre del procesador sobre el cual fue invocada la función.

- `int MPI_Init(int *argc, char **argv)`

Inicializa el ambiente, por lo que debe ser invocada antes que cualquier otra rutina MPI.

- `int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)`

Recibe a lo sumo `count` datos de tipo `datatype` en el buffer `buf`. El mensaje debe provenir del proceso `source` (excepto que se indique MPI_ANY_SOURCE) y el valor de tag debe coincidir (excepto que se indique MPI_ANY_TAG). En la estructura `status` queda el resultado de la operación.

- `int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest,int tag, MPI_Comm comm)`

Envía al proceso `dest`, `count` datos de tipo `datatype`, localizados en el buffer `buf`.

- `double MPI_Wtime(void)`

Retorna un número de segundos que representa el tiempo transcurrido desde un momento determinado en el pasado.

APENDICE B: REFERENCIA DEL FORMATO DE IMAGEN PGM (Portable graymap file format)

El formato "portable graymap" es el mínimo común denominador de los formatos de archivo de escala de grises. La definición es como sigue:

- Un "número mágico" para identificar el tipo de archivo. Un número mágico para un archivo *pgm* es el par de caracteres "P2".
- Espacios blancos (blancos, TABs, CRs, LFs)
- Un ancho W, formado por caracteres ASCII en decimal
- Blancos
- Una altura H, nuevamente en ASCII decimal
- Blancos
- El máximo valor de gris, nuevamente en ASCII decimal
- Blancos
- W * H valores de gris, cada uno en ASCII decimal, entre '0' y el valor máximo especificado, separados por blancos, empezando la esquina superior izquierda del mapa de grises, procediendo en el orden normal de lectura. El caracter '0' significa negro, y el máximo valor significa blanco
- Los caracteres desde un "#" hasta el próximo fin de línea son ignorados (se consideran comentarios)
- Ninguna línea debe ser de más de 70 caracteres.

Este es un ejemplo de un pequeño mapa de grises en este formato:

```
P2
# hola.pgm
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  0  0  3  0  0  7  7  7  7  0  0 11  0  0  0  0  0 15 15 15 15  0
0  3  0  0  3  0  0  7  0  0  7  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  3  0  0  7  0  0  7  0  0 11  0  0  0  0  0 15 15 15 15  0
0  3  0  0  3  0  0  7  0  0  7  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  0  0  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0 15  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Los programas que lean este formato deberían ser tan indulgentes como les sea posible, aceptando todo aquello que luzca remotamente como un mapa de grises.

También hay una variante del formato, que difiere en la siguiente manera:

- El "número mágico" es "P5" en lugar de "P2"
- Los valores de gris son almacenados con bytes planos en lugar de ASCII decimal
- En la sección de grises no se permiten espacios en blanco
- Después del valor máximo, sólo se permite un único caracter blanco típicamente un caracter de nueva línea)

Los archivo son más chicos y varias veces más rápidos de leer y escribir.

Notar que este formato puede ser utilizado únicamente para valores máximos menores o iguales que 255.

APENDICE C: ESTRUCTURAS DE DATOS USADAS EN LA IMPLEMENTACION

Las siguientes son las definiciones de las principales estructuras de datos utilizadas en las implementaciones secuencial y paralela.

1. Características de una imagen *pgm*. Estos son los datos que se leen del encabezado de las imágenes:

```
typedef struct pgmdef {
   int columnas,filas;       // columnas y filas
   gray maxval;              // máximo nivel de gris
   int format;               // formato del archivo pgm
   }
```

2. Descripción de un objeto. Para cada objeto se mantiene la siguiente estructura:

```
typedef struct objeto{
  int intensidad;        // valor de gris promedio
  int fila,columna;      // posición del objeto en la imagen
  int area;              // área ocupada (cantidad de pixels)
  int tieneBordeI;       // limita a la izquierda (1:Si; 0:No)
  int tieneBordeD;       // limita a la derecha (1:Si; 0:No)
}
```

Los últimos dos datos son válidos únicamente para la solución paralela.
La estructura se usa tanto para mantener los datos en memoria, como para guardar la información en el archivo de objetos correspondiente.

3. Definición del archivo de etiquetas. El archivo de etiquetas guarda, para cada punto de una imagen, un '0' si en esa posición de la imagen no hay objetos presentes, o el valor de la etiqueta asignada en caso contrario. En memoria se mantiene para el archivo de etiquetas una estructura cuya definición es la siguiente:

```
typedef struct eFILE{
  FILE *fp;              // Puntero al archivo de etiquetas
  int filas,columnas;    // Cantidad de filas y columnas de la
}                        // imagen correspondiente
```

APENDICE D: FUNCIONES AUXILIARES PARA EL SEGUIMIENTO DE TRAYECTORIAS

A continuación se transcriben las principales funciones que se utilizaron para determinar las características de una trayectoria.

1. Función que determina si un punto es falso o "fantasma"

```
int esFalso(PIXEL p) {
  return((p.fila<0)||(p.colu<0));
  }
```

2. Función que mide la distancia entre dos puntos de una imagen

```
int dist(PIXEL p1,PIXEL p2) {
   int res;

   if (esFalso(p1)||esFalso(p2))
     res=DMAX;
   else
     res=abs(p1.fila-p2.fila)+abs(p1.colu-p2.colu);
   return (res);
}
```

3. Función que calcula la longitud de un vector

```
double magnitud(int x, int y) {
  double res;
  res=sqrt(x*x + y*y);
  return(res);
}
```

4. Función que calcula el producto escalar entre dos vectores (x,y)

```
double producto(int x1,int y1,int x2,int y2) {
  double res;
  res=x1 * x2 + y1 * y2;
  return(res);
}
```

5. Función que calcula la desviación del camino dados tres puntos correspondientes a tres imágenes consecutivas

```
double desvia(PIXEL p1,PIXEL p2, PIXEL p3) {

  PIXEL v1,v2;
  double res;

  double coherDir;  // Coherencia en la dirección
```

```
  double coherVel;  // Coherencia en la velocidad
  double m1, m2;

  if (esFalso(p1))
    return(0);  // El valor mínimo (no hay desviación)

  if (esFalso(p2)||esFalso(p3))
    return(1);  // El valor máximo, porque esta faltando un punto

  v1.fila=p2.fila-p1.fila;  v1.colu=p2.colu-p1.colu;
  v2.fila=p3.fila-p2.fila;  v2.colu=p3.colu-p2.colu;
  m1=magnitud(v1.fila,v1.colu);
  if (m1==0) m1=0.00001; // evita división por cero
  m2=magnitud(v2.fila,v2.colu);
  if (m2==0) m2=0.00001; // evita división por cero

  coherDir=0.50 * (1- (producto(v1.fila, v1.colu, v2.fila, v2.colu)
/(m1*m2)));

  coherVel=0.50 * (1 - 2 *(sqrt(m1 * m2) / (m1 + m2)));

  return(coherDir+coherVel);
}
```

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

APENDICE E: Pruebas realizadas y resultados de la evaluación

Prueba Nº 1
Descripción: Un objeto pequeño desplazándose rápidamente de izquierda a derecha
Cantidad de imágenes: 13

RESULTADOS DE LA PRUEBA

A. **Realce de imágenes**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	8.597590	*0.661353*	42.126535	*3.240503*
2	8.989170	*0.691475*	32.248815	*2.480678*
3	10.819583	*0.832276*	36.084546	*2.775734*

B. **Reconocimiento de objetos**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	4.472300	*0.344023*	74.199093	*5.707623*
2	14.341866	*1.103220*	61.920869	*4.763144*
3	15.476303	*1.190485*	42.861876	*3.297067*

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	15.880627	*1.221587*	115.442883	*8.880222*
2	24.430720	*1.879286*	102.552633	*7.888664*
3	25.978525	*1.998348*	78.320218	*6.024632*

PRUEBA N°1 – RESUMEN DE RESULTADOS

A. **Realce de Imagen**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.661353*	*Tiempo:*	*3.240503*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.691475*	*Tiempo:*	*2.480678*
	Speed-up:	0.956439	Speed-up:	1.306297
	Eficiencia:	47.82%	Eficiencia:	65.31%
3	*Tiempo:*	*0.832276*	*Tiempo:*	*2.775734*
	Speed-up:	0.794632	Speed-up:	1.167440
	Eficiencia:	26.49%	Eficiencia:	38.91%

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.344023*	*Tiempo:*	*5.707623*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.103220*	*Tiempo:*	*4.763144*
	Speed-up:	0.311835	Speed-up:	1.198289
	Eficiencia:	15.59%	Eficiencia:	59.91%
3	*Tiempo:*	*1.190485*	*Tiempo:*	*3.297067*
	Speed-up:	0.288977	Speed-up:	1.731121
	Eficiencia:	9.63%	Eficiencia:	57.70%

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*1.221587*	*Tiempo:*	*8.880222*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.879286*	*Tiempo:*	*7.888664*
	Speed-up:	0.650027	Speed-up:	1.125694
	Eficiencia:	32.50%	Eficiencia:	56.28%
3	*Tiempo:*	*1.998348*	*Tiempo:*	*6.024632*
	Speed-up:	0.611298	Speed-up:	1.473986
	Eficiencia:	20.38%	Eficiencia:	49.13%

Prueba N° 2
Descripción: Varios objetos pequeños desplazándose rápidamente de derecha a izquierda
Cantidad de imágenes: 46

RESULTADOS DE LA PRUEBA

A. **Realce de imágenes**

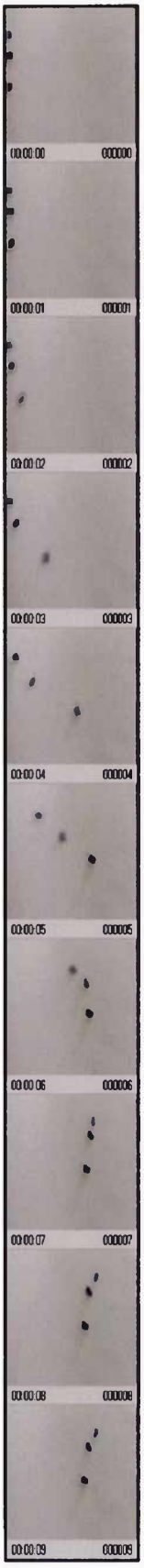
Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	30.853856	*0.670736*	147.927350	*3.215812*
2	31.903250	*0.693549*	119.672333	*2.601572*
3	38.209573	*0.830643*	127.980347	*2.782181*

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	27.716068	*0.602523*	263.302920	*5.723977*
2	52.918461	*1.150401*	211.357227	*4.594722*
3	58.671015	*1.275457*	156.967133	*3.412329*

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	75.168066	*1.634088*	405.472456	*8.814619*
2	89.647865	*1.948867*	384.103304	*8.350072*
3	97.050162	*2.109786*	285.041391	*6.196552*



PRUEBA Nº 2 – RESUMEN DE RESULTADOS

A. **Realce de Imagen**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.670736*	*Tiempo:*	*3.215812*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.693549*	*Tiempo:*	*2.601572*
	Speed-up:	0.967107	Speed-up:	1.236103
	Eficiencia:	48.36%	Eficiencia:	61.81%
3	*Tiempo:*	*0.830643*	*Tiempo:*	*2.782181*
	Speed-up:	0.807490	Speed-up:	1.155860
	Eficiencia:	26.92%	Eficiencia:	38.53%

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.602523*	*Tiempo:*	*5.723977*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.150401*	*Tiempo:*	*4.594722*
	Speed-up:	0.523750	Speed-up:	1.245772
	Eficiencia:	26.19%	Eficiencia:	62.29%
3	*Tiempo:*	*1.275457*	*Tiempo:*	*3.412329*
	Speed-up:	0.472398	Speed-up:	1.677440
	Eficiencia:	15.75%	Eficiencia:	55.91%

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*1.634088*	*Tiempo:*	*8.814619*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.948867*	*Tiempo:*	*8.350072*
	Speed-up:	0.838481	Speed-up:	1.055634
	Eficiencia:	41.92%	Eficiencia:	52.78%
3	*Tiempo:*	*2.109786*	*Tiempo:*	*6.196552*
	Speed-up:	0.774528	Speed-up:	1.422504
	Eficiencia:	25.82%	Eficiencia:	47.42%

Prueba N° 3
Descripción: Varios objetos pequeños desplazándose a diferentes velocidades
Cantidad de imágenes: 33

RESULTADOS DE LA PRUEBA

A. **Realce de imágenes**

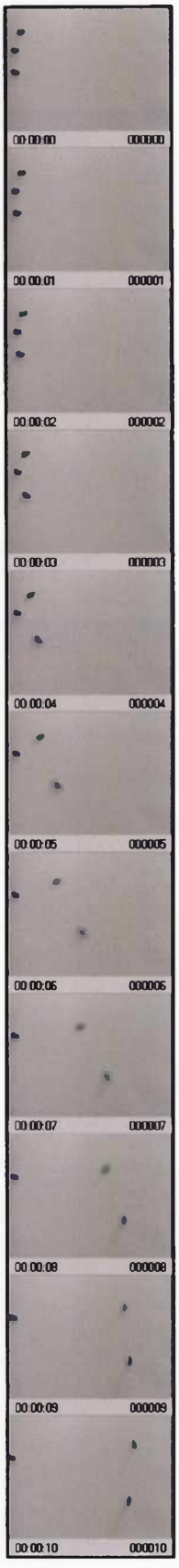
Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	20.063298	*0.607979*	105.637810	*3.201146*
2	22.007735	*0.666901*	86.833161	*2.631308*
3	27.303314	*0.827373*	91.886912	*2.784452*

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	18.460761	*0.559417*	180.207708	*5.460840*
2	38.835198	*1.176824*	152.059303	*4.607858*
3	40.316709	*1.221718*	128.391505	*3.890652*

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	50.891561	*1.542169*	284.071690	*8.608233*
2	64.698676	*1.960566*	238.066196	*7.214127*
3	69.616327	*2.109586*	219.458026	*6.650243*



PRUEBA Nº 3 – RESUMEN DE RESULTADOS

A. Realce de Imagen

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.607979*	*Tiempo:*	*3.201146*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.666901*	*Tiempo:*	*2.631308*
	Speed-up:	0.911648	Speed-up:	1.216561
	Eficiencia:	45.58%	Eficiencia:	60.83%
3	*Tiempo:*	*0.827373*	*Tiempo:*	*2.784452*
	Speed-up:	0.734830	Speed-up:	1.149650
	Eficiencia:	24.49%	Eficiencia:	38.32%

B. Reconocimiento de Objetos

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.559417*	*Tiempo:*	*5.460840*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.176824*	*Tiempo:*	*4.607858*
	Speed-up:	0.475362	Speed-up:	1.185115
	Eficiencia:	23.77%	Eficiencia:	59.26%
3	*Tiempo:*	*1.221718*	*Tiempo:*	*3.890652*
	Speed-up:	0.457894	Speed-up:	1.403580
	Eficiencia:	15.26%	Eficiencia:	46.79%

C. Seguimiento de Trayectorias

Procesos	320x240		720 x 540	
1	*Tiempo:*	*1.542169*	*Tiempo:*	*8.608233*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.960566*	*Tiempo:*	*7.214127*
	Speed-up:	0.786594	Speed-up:	1.193247
	Eficiencia:	39.33%	Eficiencia:	59.66%
3	*Tiempo:*	*2.109586*	*Tiempo:*	*6.650243*
	Speed-up:	0.731029	Speed-up:	1.294424
	Eficiencia:	24.37%	Eficiencia:	43.15%

Prueba N° 4
Descripción: Varios objetos pequeños desplazándose a velocidad moderada
Cantidad de imágenes: 24

RESULTADOS DE LA PRUEBA

A. Realce de imágenes

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	15.882770	*0.661782*	77.142549	*3.214273*
2	16.306605	*0.679442*	63.091425	*2.628809*
3	20.194209	*0.841425*	67.421560	*2.809232*

B. Reconocimiento de Objetos

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	10.398260	*0.433261*	139.993901	*5.833079*
2	27.013397	*1.125558*	115.932653	*4.830527*
3	29.545668	*1.231070*	81.903233	*3.412635*

C. Seguimiento de Trayectorias

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	34.402317	*1.433430*	214.242276	*8.926761*
2	47.156956	*1.964873*	186.179824	*7.757493*
3	50.849405	*2.118725*	144.156657	*6.006527*

PRUEBA Nº 4 – RESUMEN DE RESULTADOS

A. **Realce de Imagen**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.661782*	*Tiempo:*	*3.214273*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.679442*	*Tiempo:*	*2.628809*
	Speed-up:	0.974008	Speed-up:	1.222711
	Eficiencia:	48.70%	Eficiencia:	61.14%
3	*Tiempo:*	*0.841425*	*Tiempo:*	*2.809232*
	Speed-up:	0.786501	Speed-up:	1.144182
	Eficiencia:	26.22%	Eficiencia:	38.14%

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.433261*	*Tiempo:*	*5.833079*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.125558*	*Tiempo:*	*4.830527*
	Speed-up:	0.384930	Speed-up:	1.207545
	Eficiencia:	19.25%	Eficiencia:	60.38%
3	*Tiempo:*	*1.231070*	*Tiempo:*	*3.412635*
	Speed-up:	0.351939	Speed-up:	1.709260
	Eficiencia:	11.73%	Eficiencia:	56.98%

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*1.433430*	*Tiempo:*	*8.926761*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.964873*	*Tiempo:*	*7.757493*
	Speed-up:	0.729528	Speed-up:	1.150728
	Eficiencia:	36.48%	Eficiencia:	57.54%
3	*Tiempo:*	*2.118725*	*Tiempo:*	*6.006527*
	Speed-up:	0.676553	Speed-up:	1.486177
	Eficiencia:	22.55%	Eficiencia:	49.54%

Prueba Nº 5
Descripción: Varios objetos pequeños desplazándose a velocidad moderada
Cantidad de imágenes: 28

RESULTADOS DE LA PRUEBA

A. **Realce de imágenes**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	19.459920	*0.694997*	92.855396	*3.316264*
2	19.603024	*0.700108*	75.105607	*2.682343*
3	23.788871	*0.849603*	78.839334	*2.815691*

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	18.599148	*0.664255*	212.700467	*7.596445*
2	32.261095	*1.152182*	168.203469	*6.007267*
3	36.618054	*1.307788*	123.556490	*4.412732*

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	47.842296	*1.708653*	303.754823	*10.848387*
2	53.680784	*1.917171*	243.289912	*8.688925*
3	56.829157	*2.029613*	203.394793	*7.264100*

PRUEBA Nº 5 – RESUMEN DE RESULTADOS

A. Realce de Imagen

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.694997*	*Tiempo:*	*3.316264*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.700108*	*Tiempo:*	*2.682343*
	Speed-up:	0.992700	Speed-up:	1.236331
	Eficiencia:	49.63%	Eficiencia:	61.82%
3	*Tiempo:*	*0.849603*	*Tiempo:*	*2.815691*
	Speed-up:	0.818026	Speed-up:	1.177780
	Eficiencia:	27.27%	Eficiencia:	39.26%

B. Reconocimiento de Objetos

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.664255*	*Tiempo:*	*7.596445*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.152182*	*Tiempo:*	*6.007267*
	Speed-up:	0.576519	Speed-up:	1.264543
	Eficiencia:	28.83%	Eficiencia:	63.23%
3	*Tiempo:*	*1.307788*	*Tiempo:*	*4.412732*
	Speed-up:	0.507923	Speed-up:	1.721484
	Eficiencia:	16.93%	Eficiencia:	57.38%

C. Seguimiento de Trayectorias

Procesos	320x240		720 x 540	
1	*Tiempo:*	*1.708653*	*Tiempo:*	*10.848387*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.917171*	*Tiempo:*	*8.688925*
	Speed-up:	0.891237	Speed-up:	1.248530
	Eficiencia:	44.56%	Eficiencia:	62.43%
3	*Tiempo:*	*2.029613*	*Tiempo:*	*7.264100*
	Speed-up:	0.841862	Speed-up:	1.493425
	Eficiencia:	28.06%	Eficiencia:	49.78%

Prueba N° 6
Descripción: Varios objetos pequeños desplazándose rápidamente de arriba hacia abajo
Cantidad de imágenes: 15

RESULTADOS DE LA PRUEBA

A. **Realce de imágenes**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	9.966897	*0.664460*	45.734730	*3.048982*
2	10.414866	*0.694324*	38.864098	*2.590940*
3	12.256668	*0.817111*	40.066896	*2.671126*

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	14.664796	*0.977653*	108.336183	*7.222412*
2	18.625797	*1.241720*	81.171825	*5.411455*
3	18.479737	*1.231982*	59.524298	*3.968287*

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	34.451748	*2.296783*	156.604176	*10.440278*
2	31.012501	*2.067500*	116.996847	*7.799790*
3	32.986934	*2.199129*	103.572098	*6.904807*

PRUEBA Nº 6 – RESUMEN DE RESULTADOS

A. **Realce de Imagen**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.664460*	*Tiempo:*	*3.048982*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.694324*	*Tiempo:*	*2.590940*
	Speed-up:	0.956987	Speed-up:	1.176786
	Eficiencia:	47.85%	Eficiencia:	58.84%
3	*Tiempo:*	*0.817111*	*Tiempo:*	*2.671126*
	Speed-up:	0.813182	Speed-up:	1.141459
	Eficiencia:	27.11%	Eficiencia:	38.05%

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.977653*	*Tiempo:*	*7.222412*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.241720*	*Tiempo:*	*5.411455*
	Speed-up:	0.787338	Speed-up:	1.334653
	Eficiencia:	39.37%	Eficiencia:	66.73%
3	*Tiempo:*	*1.231982*	*Tiempo:*	*3.968287*
	Speed-up:	0.793561	Speed-up:	1.820033
	Eficiencia:	26.45%	Eficiencia:	60.67%

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*2.296783*	*Tiempo:*	*10.440278*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.067500*	*Tiempo:*	*7.799790*
	Speed-up:	1.110899	Speed-up:	1.338533
	Eficiencia:	55.54%	Eficiencia:	66.93%
3	*Tiempo:*	*2.199129*	*Tiempo:*	*6.904807*
	Speed-up:	1.044406	Speed-up:	1.512031
	Eficiencia:	34.81%	Eficiencia:	50.40%

Prueba N° 7
Descripción: Un objeto mediano desplazándose muy despacio de izquierda a derecha
Cantidad de imágenes: 135

RESULTADOS DE LA PRUEBA

A. **Realce de imágenes**

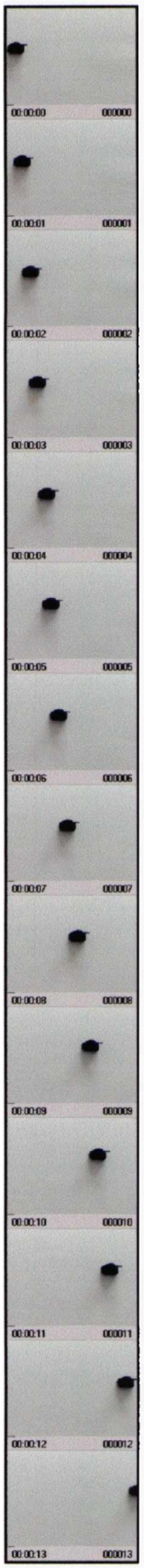
Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	92.866895	*0.687903*	437.594981	*3.241444*
2	94.826327	*0.702417*	354.658957	*2.627103*
3	114.159415	*0.845625*	379.225064	*2.809075*

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	85.702739	*0.634835*	804.699149	*5.960734*
2	172.432971	*1.277281*	722.721217	*5.353490*
3	179.678824	*1.330954*	565.360532	*4.187856*

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	235.767273	*1.746424*	1342.456614	*9.944123*
2	283.800169	*2.102223*	1108.826652	*8.213531*
3	292.351170	*2.165564*	996.933895	*7.384696*



PRUEBA Nº 7 – RESUMEN DE RESULTADOS

A. **Realce de Imagen**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.687903*	*Tiempo:*	*3.241444*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.702417*	*Tiempo:*	*2.627103*
	Speed-up:	0.979337	Speed-up:	1.233847
	Eficiencia:	48.97%	Eficiencia:	61.69%
3	*Tiempo:*	*0.845625*	*Tiempo:*	*2.809075*
	Speed-up:	0.813484	Speed-up:	1.153919
	Eficiencia:	27.12%	Eficiencia:	38.46%

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.634835*	*Tiempo:*	*5.960734*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.277281*	*Tiempo:*	*5.353490*
	Speed-up:	0.497021	Speed-up:	1.113430
	Eficiencia:	24.85%	Eficiencia:	55.67%
3	*Tiempo:*	*1.330954*	*Tiempo:*	*4.187856*
	Speed-up:	0.476977	Speed-up:	1.423338
	Eficiencia:	15.90%	Eficiencia:	47.44%

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*1.746424*	*Tiempo:*	*9.944123*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.102223*	*Tiempo:*	*8.213531*
	Speed-up:	0.830751	Speed-up:	1.210700
	Eficiencia:	41.54%	Eficiencia:	60.54%
3	*Tiempo:*	*2.165564*	*Tiempo:*	*7.384696*
	Speed-up:	0.806452	Speed-up:	1.346585
	Eficiencia:	26.88%	Eficiencia:	44.89%

Prueba N° 8
Descripción: Dos objetos medianos desplazándose lentamente en dirección diagonal
Cantidad de imágenes: 154

RESULTADOS DE LA PRUEBA

A. **Realce de imágenes**

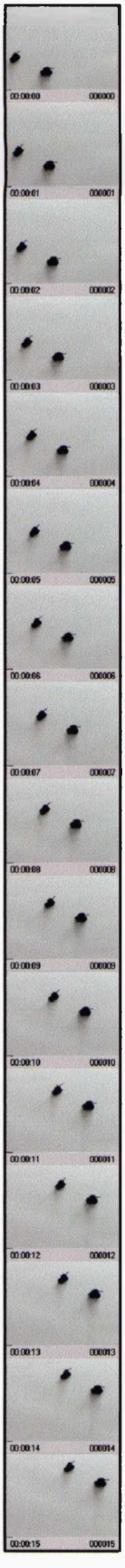
Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	106.122490	*0.689107*	501.771751	*3.258258*
2	107.829722	*0.700193*	405.188118	*2.631092*
3	129.907155	*0.843553*	434.256983	*2.819851*

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	175.964123	*1.142624*	1724.242837	*11.196382*
2	211.368919	*1.372525*	1114.097999	*7.234403*
3	207.055506	*1.344516*	767.426007	*4.983286*

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	384.168021	*2.494598*	2404.095754	*15.611011*
2	346.684599	*2.251199*	1549.671839	*10.062804*
3	332.603723	*2.159764*	1273.783969	*8.271324*



PRUEBA Nº 8 – RESUMEN DE RESULTADOS

A. Realce de Imagen

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.689107*	*Tiempo:*	*3.258258*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.700193*	*Tiempo:*	*2.631092*
	Speed-up:	0.984167	Speed-up:	1.238367
	Eficiencia:	49.21%	Eficiencia:	61.92%
3	*Tiempo:*	*0.843553*	*Tiempo:*	*2.819851*
	Speed-up:	0.816910	Speed-up:	1.155472
	Eficiencia:	27.23%	Eficiencia:	38.52%

B. Reconocimiento de Objetos

Procesos	320x240		720 x 540	
1	*Tiempo:*	*1.142624*	*Tiempo:*	*11.196382*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.372525*	*Tiempo:*	*7.234403*
	Speed-up:	0.832498	Speed-up:	1.547658
	Eficiencia:	41.62%	Eficiencia:	77.38%
3	*Tiempo:*	*1.344516*	*Tiempo:*	*4.983286*
	Speed-up:	0.849840	Speed-up:	2.246787
	Eficiencia:	28.33%	Eficiencia:	74.89%

C. Seguimiento de Trayectorias

Procesos	320x240		720 x 540	
1	*Tiempo:*	*2.494598*	*Tiempo:*	*15.611011*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.251199*	*Tiempo:*	*10.062804*
	Speed-up:	1.108120	Speed-up:	1.551358
	Eficiencia:	55.41%	Eficiencia:	77.57%
3	*Tiempo:*	*2.159764*	*Tiempo:*	*8.271324*
	Speed-up:	1.155032	Speed-up:	1.887365
	Eficiencia:	38.50%	Eficiencia:	62.91%

Prueba Nº 9
Descripción: Un objeto grande desplazándose lentamente de izquierda a derecha
Cantidad de imágenes: 90

RESULTADOS DE LA PRUEBA

A. **Realce de imágenes**

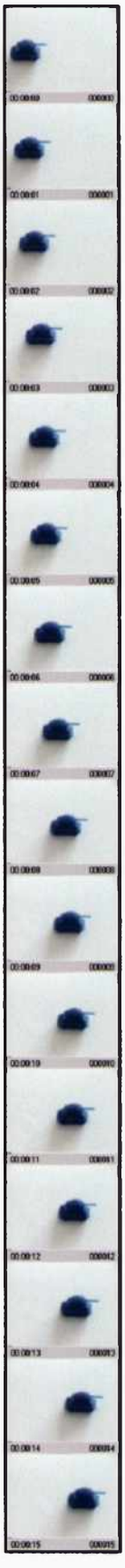
Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	62.743466	*0.697150*	298.921943	*3.321355*
2	62.674332	*0.696381*	241.875356	*2.687504*
3	76.724294	*0.852492*	254.111393	*2.823460*

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	131.084138	*1.456490*	1161.717307	*12.907970*
2	160.514166	*1.783491*	826.526653	*9.183629*
3	129.650579	*1.440562*	493.959242	*5.488436*

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	286.693169	*3.185480*	1457.824220	*16.198047*
2	234.644310	*2.607159*	1080.709628	*12.007885*
3	203.114865	*2.256832*	741.955573	*8.243951*



PRUEBA Nº 9 – RESUMEN DE RESULTADOS

A. **Realce de Imagen**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.697150*	*Tiempo:*	*3.321355*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.696381*	*Tiempo:*	*2.687504*
	Speed-up:	1.001103	Speed-up:	1.235851
	Eficiencia:	50.06%	Eficiencia:	61.79%
3	*Tiempo:*	*0.852492*	*Tiempo:*	*2.823460*
	Speed-up:	0.817778	Speed-up:	1.176342
	Eficiencia:	27.26%	Eficiencia:	39.21%

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*1.456490*	*Tiempo:*	*12.907970*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*1.783491*	*Tiempo:*	*9.183629*
	Speed-up:	0.816652	Speed-up:	1.405541
	Eficiencia:	40.83%	Eficiencia:	70.28%
3	*Tiempo:*	*1.440562*	*Tiempo:*	*5.488436*
	Speed-up:	1.011057	Speed-up:	2.351849
	Eficiencia:	33.70%	Eficiencia:	78.39%

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*3.185480*	*Tiempo:*	*16.198047*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.607159*	*Tiempo:*	*12.007885*
	Speed-up:	1.221820	Speed-up:	1.348951
	Eficiencia:	61.09%	Eficiencia:	67.45%
3	*Tiempo:*	*2.256832*	*Tiempo:*	*8.243951*
	Speed-up:	1.411483	Speed-up:	1.964840
	Eficiencia:	47.05%	Eficiencia:	65.49%

Prueba N° 10
Descripción: Dos objetos grandes desplazándose lentamente de izquierda a derecha
Cantidad de imágenes: 67

RESULTADOS DE LA PRUEBA

A. Realce de imágenes

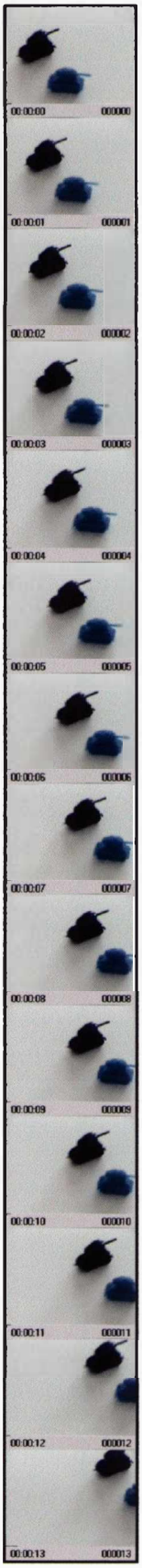
Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	45.635085	*0.681121*	218.770475	*3.265231*
2	46.276942	*0.690701*	177.716391	*2.652483*
3	56.701072	*0.846285*	187.864024	*2.803941*

B. Reconocimiento de Objetos

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	230.433956	*3.439313*	2095.414786	*31.274848*
2	198.174653	*2.957831*	1383.151174	*20.644047*
3	136.522385	*2.037648*	780.180151	*11.644480*

C. Seguimiento de Trayectorias

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	441.653803	*6.591848*	2310.115188	*34.479331*
2	254.146891	*3.793237*	1447.931087	*21.610912*
3	193.061261	*2.881511*	960.165031	*14.330821*



PRUEBA Nº 10 – RESUMEN DE RESULTADOS

A. **Realce de Imagen**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.681121*	*Tiempo:*	*3.265231*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.690701*	*Tiempo:*	*2.652483*
	Speed-up:	0.986130	Speed-up:	1.231009
	Eficiencia:	49.31%	Eficiencia:	61.55%
3	*Tiempo:*	*0.846285*	*Tiempo:*	*2.803941*
	Speed-up:	0.804836	Speed-up:	1.164515
	Eficiencia:	26.83%	Eficiencia:	38.82%

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*3.439313*	*Tiempo:*	*31.274848*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.957831*	*Tiempo:*	*20.644047*
	Speed-up:	1.162782	Speed-up:	1.514957
	Eficiencia:	58.14%	Eficiencia:	75.75%
3	*Tiempo:*	*2.037648*	*Tiempo:*	*11.644480*
	Speed-up:	1.687884	Speed-up:	2.685809
	Eficiencia:	56.26%	Eficiencia:	89.53%

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*6.591848*	*Tiempo:*	*34.479331*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*3.793237*	*Tiempo:*	*21.610912*
	Speed-up:	1.737790	Speed-up:	1.595459
	Eficiencia:	86.89%	Eficiencia:	79.77%
3	*Tiempo:*	*2.881511*	*Tiempo:*	*14.330821*
	Speed-up:	2.287636	Speed-up:	2.405956
	Eficiencia:	76.25%	Eficiencia:	80.20%

Prueba N° 11
Descripción: Dos objetos grandes desplazándose rápidamente en dirección diagonal
Cantidad de imágenes: 32

RESULTADOS DE LA PRUEBA

A. **Realce de imágenes**

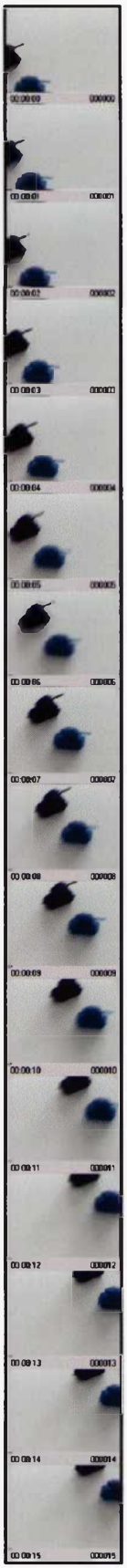
Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	21.558851	*0.673714*	102.231186	*3.194725*
2	22.137104	*0.691785*	85.636805	*2.676150*
3	27.146039	*0.848314*	88.867290	*2.777103*

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	91.504661	*2.859521*	807.318177	*25.228693*
2	85.175665	*2.661740*	587.390724	*18.355960*
3	67.897146	*2.121786*	385.804443	*12.056389*

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	179.753936	*5.617311*	919.252572	*28.726643*
2	111.719658	*3.491239*	682.806239	*21.337695*
3	93.049857	*2.907808*	473.445875	*14.795184*



PRUEBA Nº 11 – RESUMEN DE RESULTADOS

A. Realce de Imagen

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.673714*	*Tiempo:*	*3.194725*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.691785*	*Tiempo:*	*2.676150*
	Speed-up:	0.973879	Speed-up:	1.193776
	Eficiencia:	48.69%	Eficiencia:	59.69%
3	*Tiempo:*	*0.848314*	*Tiempo:*	*2.777103*
	Speed-up:	0.794180	Speed-up:	1.150380
	Eficiencia:	26.47%	Eficiencia:	38.35%

B. Reconocimiento de Objetos

Procesos	320x240		720 x 540	
1	*Tiempo:*	*2.859521*	*Tiempo:*	*25.228693*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.661740*	*Tiempo:*	*18.355960*
	Speed-up:	1.074305	Speed-up:	1.374414
	Eficiencia:	53.72%	Eficiencia:	68.72%
3	*Tiempo:*	*2.121786*	*Tiempo:*	*12.056389*
	Speed-up:	1.347695	Speed-up:	2.092558
	Eficiencia:	44.92%	Eficiencia:	69.75%

C. Seguimiento de Trayectorias

Procesos	320x240		720 x 540	
1	*Tiempo:*	*5.617311*	*Tiempo:*	*28.726643*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*3.491239*	*Tiempo:*	*21.337695*
	Speed-up:	1.608973	Speed-up:	1.346286
	Eficiencia:	80.45%	Eficiencia:	67.31%
3	*Tiempo:*	*2.907808*	*Tiempo:*	*14.795184*
	Speed-up:	1.931802	Speed-up:	1.941621
	Eficiencia:	64.39%	Eficiencia:	64.72%

Prueba N° 12
Descripción: Dos objetos grandes desplazándose en varios sentidos a velocidad moderada
Cantidad de imágenes: 51

RESULTADOS DE LA PRUEBA

A. Realce de imágenes

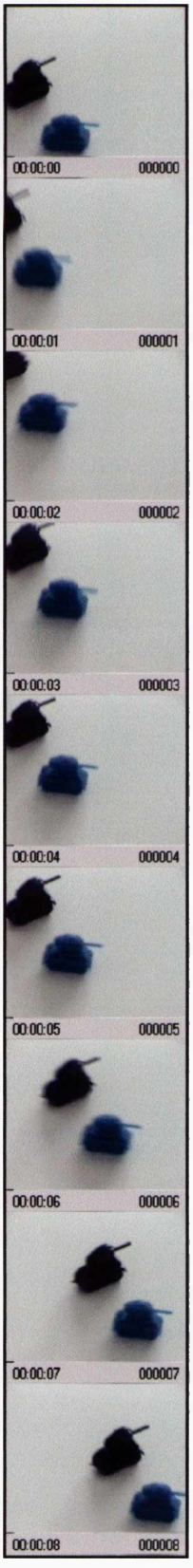
Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	34.627664	*0.678974*	166.823270	*3.271045*
2	35.507460	*0.696225*	135.491579	*2.656698*
3	43.287557	*0.848776*	143.760772	*2.818839*

B. Reconocimiento de Objetos

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	154.773564	*3.034776*	1328.835640	*26.055601*
2	144.545759	*2.834231*	951.508914	*18.657038*
3	117.478175	*2.303494*	652.842981	*12.800843*

C. Seguimiento de Trayectorias

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	298.282909	*5.848684*	1492.712601	*29.268875*
2	188.128028	*3.688785*	1102.352092	*21.614747*
3	159.520915	*3.127861*	785.680458	*15.405499*



PRUEBA N° 12 – RESUMEN DE RESULTADOS

A. **Realce de Imagen**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.678974*	*Tiempo:*	*3.271045*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.696225*	*Tiempo:*	*2.656698*
	Speed-up:	0.975222	Speed-up:	1.231245
	Eficiencia:	48.76%	Eficiencia:	61.56%
3	*Tiempo:*	*0.848776*	*Tiempo:*	*2.818839*
	Speed-up:	0.799945	Speed-up:	1.160423
	Eficiencia:	26.66%	Eficiencia:	38.68%

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*3.034776*	*Tiempo:*	*26.055601*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.834231*	*Tiempo:*	*18.657038*
	Speed-up:	1.070758	Speed-up:	1.396556
	Eficiencia:	53.54%	Eficiencia:	69.83%
3	*Tiempo:*	*2.303494*	*Tiempo:*	*12.800843*
	Speed-up:	1.317467	Speed-up:	2.035460
	Eficiencia:	43.92%	Eficiencia:	67.85%

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*5.848684*	*Tiempo:*	*29.268875*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*3.688785*	*Tiempo:*	*21.614747*
	Speed-up:	1.585531	Speed-up:	1.354116
	Eficiencia:	79.28%	Eficiencia:	67.71%
3	*Tiempo:*	*3.127861*	*Tiempo:*	*15.405499*
	Speed-up:	1.869867	Speed-up:	1.899898
	Eficiencia:	62.33%	Eficiencia:	63.33%

Prueba N° 13
Descripción: Varios objetos grandes desplazándose diagonalmente a velocidad moderada
Cantidad de imágenes: 49

RESULTADOS DE LA PRUEBA

A. **Realce de imágenes**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	33.370801	*0.681037*	160.196203	*3.269310*
2	33.950492	*0.692867*	129.914826	*2.651323*
3	40.718194	*0.830984*	137.505244	*2.806229*

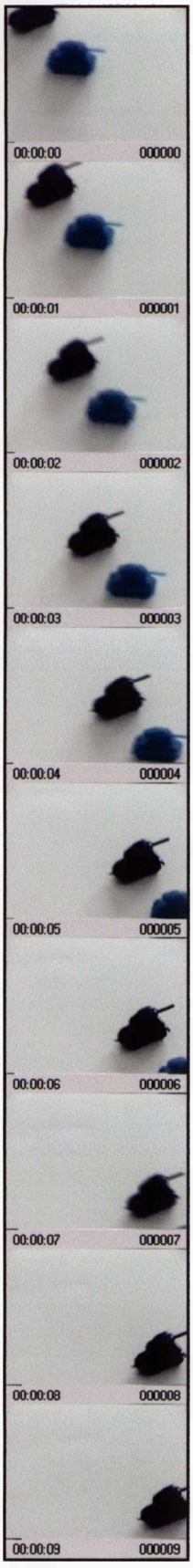
B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	153.844449	*3.139683*	1323.192539	*27.003929*
2	138.799819	*2.832649*	884.382105	*18.048614*
3	108.870379	*2.221844*	599.306227	*12.230739*

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
	Tiempo total	Tiempo x imagen	Tiempo total	Tiempo x imagen
1	294.119349	*6.002436*	1484.908778	*30.304261*
2	176.749222	*3.607127*	1027.069656	*20.960605*
3	148.244909	*3.025406*	725.146032	*14.798899*

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.



PRUEBA Nº 13 – RESUMEN DE RESULTADOS

A. **Realce de Imagen**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*0.681037*	*Tiempo:*	*3.269310*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*0.692867*	*Tiempo:*	*2.651323*
	Speed-up:	0.982925	Speed-up:	1.233086
	Eficiencia:	49.15%	Eficiencia:	61.65%
3	*Tiempo:*	*0.830984*	*Tiempo:*	*2.806229*
	Speed-up:	0.819555	Speed-up:	1.165019
	Eficiencia:	27.32%	Eficiencia:	38.83%

B. **Reconocimiento de Objetos**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*3.139683*	*Tiempo:*	*27.003929*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*2.832649*	*Tiempo:*	*18.048614*
	Speed-up:	1.108391	Speed-up:	1.496177
	Eficiencia:	55.42%	Eficiencia:	74.81%
3	*Tiempo:*	*2.221844*	*Tiempo:*	*12.230739*
	Speed-up:	1.413097	Speed-up:	2.207874
	Eficiencia:	47.10%	Eficiencia:	73.60%

C. **Seguimiento de Trayectorias**

Procesos	320x240		720 x 540	
1	*Tiempo:*	*6.002436*	*Tiempo:*	*30.304261*
	Speed-up:	1.000000	Speed-up:	1.000000
	Eficiencia:	100.00%	Eficiencia:	100.00%
2	*Tiempo:*	*3.607127*	*Tiempo:*	*20.960605*
	Speed-up:	1.664049	Speed-up:	1.445772
	Eficiencia:	83.20%	Eficiencia:	72.29%
3	*Tiempo:*	*3.025406*	*Tiempo:*	*14.798899*
	Speed-up:	1.984010	Speed-up:	2.047738
	Eficiencia:	66.13%	Eficiencia:	68.26%

APENDICE F: BIBLIOGRAFIA Y REFERENCIAS

1. Ramesh Jain. Rangachar Kasturi. Brian G. Schunck. **Machine Vision**. Mc. Graw-Hill International Editions (1995)

2. Anil K. Jain. **Fundamentals of Digital Image Processing**. Prentice Hall Information and System Sciences Series (1989)

3. H. Stephen Morse. **Practical Parallel Computing**. AP Professional.

4. Gregory R. Andrews. **Concurrent Programming Principles and Practice**. The Benjamin/Commings Publishing Company, Inc (1991)

5. Marc Snir - Steve Otto – Steven Huss-Lederman – David Walker – Jack Dongarra. **MPI: The Complete Reference.** The MIT Press (1996)

6. William Group. **Tutorial on MPI: The Message-Passing Interface**. Mathematics and Computer Science Division – Argonne National Laboratory

7. GDB/RBD. **MPI Primer / Developing with LAM**. The Ohio State University (1995)

8. Matt Welsh and Lar Kaufman. **Running LINUX**. O'Reilly & Associates, Inc (1996 – Second Edition)

9. http://www.mcs.anl.gov/pub/mpi – Información sobre implementaciones y sobre MPI en general - Argonne National Laboratory

10. http://www.erc.msstate.edu/mpi – Información sobre implementaciones y sobre MPI en general – Mississippi State University.

11. ftp://info.mcs.anl.gov/pub/mpi Implementación MPICH de Argonne National Laboratory y Mississippi State University.

12. www.acme.com/software/pbmplus Image file format conversion package.

13. http://www.perl.com

BIBLIOTECA
FAC. DE INFORMÁTICA
U.N.L.P.

DONACION..... TES
$..... 00|12
Fecha..... 6-10-05
Inv. E. Dk-39 Inv. B..... 2121