Tesina de grado

Seguimiento y estimación de trayectorias

Juan Cortabitarte 2036/4

Pablo Novarini 2179/0

1 Motivación y objetivos generales

Analizar el problema de seguimiento y estimación en tiempo real de las trayectorias de un conjunto de objetos con movimiento propio en un ambiente conocido, a partir de una secuencia de imágenes de video. Estudiar cada una de las etapas del sistema y los algoritmos que pueden aplicarse en cada una de ellas. En particular interesa realizar una implementación para un caso concreto utilizando técnicas de programación paralela con el objetivo de obtener tiempos de respuesta admisibles para el procesamiento en tiempo real.

En este trabajo se utilizan herramientas automáticas para generar secuencias de imágenes que representen el movimiento de un conjunto de objetos durante un período de tiempo dentro de una escena ficticia. El objetivo de esto es realizar pruebas en escenarios completamente conocidos (sabiendo con exactitud los recorridos de los objetos) y así poder evaluar los resultados obtenidos por el sistema en una forma precisa.

De todos modos las imágenes podrían corresponder a una escena real y ser capturadas en tiempo real para su procesamiento.

A partir de la suposición de ciertas características en el movimiento de los objetos y del análisis de los recorridos realizados por éstos hasta un determinado instante, se busca obtener predicciones de sus trayectorias futuras, para determinar posibles puntos de colisión entre los diferentes objetos de la escena.

2 Descripción general del problema

2.1 Introducción

Existe una amplia gama de aplicaciones en donde es necesario resolver el problema de seguimiento y estimación de las trayectorias realizadas por un conjunto de entidades u objetos en movimiento. Entre ellas podemos mencionar el guiado asistido de vehículos, el seguimiento de misiles mediante el uso de radares, el análisis del movimiento de espermatozoides en una muestra de semen, las competencias de fútbol robótico, etc.

En esta tesina se estudia el *seguimiento y la estimación* de las trayectorias realizadas por un conjunto de entidades con movimiento autónomo dentro de una escena.

Por *seguimiento de trayectorias* entendemos a la acción de identificar el recorrido realizado por cada uno de los objetos de la escena. Basándose en ésta información y bajo la suposición de ciertas características en el movimiento de los objetos, se realiza la *predicción de trayectorias*, que consiste en estimar el movimiento futuro de cada uno de ellos.

Se busca obtener predicciones de las trayectorias con el mínimo error, con el objetivo de determinar futuros posibles puntos de conflicto o choque entre los diferentes objetos. Dado que se requieren respuestas en tiempo real, se estudia la paralelización de los algoritmos que solucionan el problema, con el objetivo de mejorar los tiempos de respuesta del sistema.

El reconocimiento de la escena y las entidades que en ella aparecen se realiza a partir de imágenes capturadas por algún sensor óptico (cámara). La entrada al sistema es entonces una secuencia de imágenes donde cada una representa a la escena en un instante de tiempo.

Es importante tener en cuenta las características de la escena, las de la cámara y la posición de ésta con respecto a la escena. La cámara puede estar fija o en movimiento, ubicada en el mismo plano de la escena capturándola lateralmente, sobre ella en un plano paralelo a un determinado ángulo de inclinación, etc.

A continuación se describen los pasos o etapas que deben resolverse para solucionar el problema planteado (Gráfico 2-1). De este análisis surgirá una estructura general del sistema.

La entrada la constituye una secuencia de imágenes, la cual podemos suponer infinita, en el caso que las imágenes ingresen al sistema mediante un dispositivo de captura en funcionamiento permanente, como ser una cámara de video digital. Cada nueva imagen (frame) procesada por el sistema proporciona nueva información que provoca un reajuste de las estimaciones previas.

La salida es el conjunto de recorridos que se estima que realizarán los objetos encontrados en la escena.

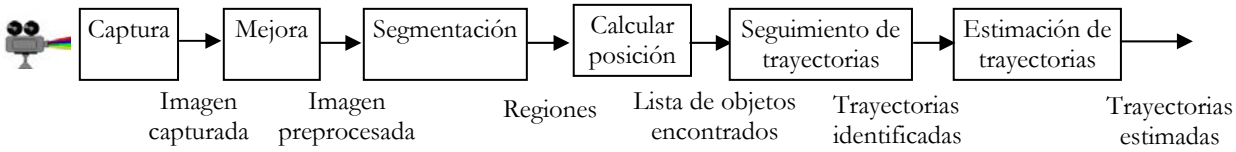


Gráfico 2-1. Etapas en la resolución del problema

Continuamente ingresa al sistema un nuevo frame de entrada, el cual debe pasar por varias etapas de procesamiento. Haremos una breve mención de cada una de las etapas.

La etapa de *mejora de la imagen* tiene como objetivo eliminar partes indeseables de la imagen obtenida mediante el dispositivo de captura así como realzar las partes interesantes de la misma. Existe un sinfín de problemas relacionados con la captura, que incluyen, entre otras cosas, ruidos, malas condiciones de iluminación, distorsiones geométricas provocadas por la lente de la cámara o debido a la perspectiva, imagen borrosa o fuera de foco. Es necesario aplicar a la imagen capturada un conjunto de algoritmos con el fin de obtener una nueva imagen que elimine o atenúe las características indeseables de la original. El *mejorado de la imagen* es la etapa donde se aplican tales algoritmos.

La *segmentación* consiste en subdividir la imagen en sus partes constituyentes u objetos. El nivel de detalle al que se llega dependerá del problema a resolver, lo que significa que la segmentación se dará por finalizada cuando los objetos de interés de la aplicación hayan sido aislados.

Una vez identificados los objetos de interés en el frame, es necesario determinar la posición donde se encuentra cada uno dentro de la escena, para lo cual debe establecerse un sistema de coordenadas. Para realizar esta tarea con éxito es importante tener en cuenta la ubicación de la cámara con respecto a la escena.

La etapa de *armado de trayectorias* utiliza la información de todos los frames procesados hasta el momento y consiste en armar un recorrido para cada objeto encontrado. Aquí es donde debe resolverse el problema de la correspondencia de objetos entre frames, es decir, determinar si objetos encontrados en frames diferentes son el mismo objeto de la escena real.

Una vez que se ha armado el recorrido realizado por cada uno de los objetos de la escena, se utiliza dicha información en la etapa de *predicción de trayectorias* para estimar el recorrido que realizará cada uno de los objetos en instantes posteriores de tiempo.

La salida de la etapa de predicción consiste en los recorridos que se estima que realizarán los objetos en un futuro de tiempo cercano. Es importante que bajo ciertas suposiciones acerca del movimiento de los objetos, se establezca algún modelo del error

cometido en las predicciones. De esta forma, podrán estimarse por ejemplo, colisiones entre objetos con una probabilidad de éxito determinada.

2.2 Consideraciones tenidas en cuenta en este trabajo

Anteriormente se presentó de una forma muy general el problema de seguimiento y estimación de trayectorias, y las etapas involucradas en su resolución. Cuando se desea resolver un problema de este tipo, hay aspectos particulares del problema concreto muy importantes que deben ser tenidos en cuenta y decisiones que deben tomarse en función a ellos, que posiblemente tengan una gran influencia en el éxito o fracaso de la solución. Como sucede en la gran mayoría de aplicaciones informáticas, una solución exitosa para un caso concreto puede resultar un fracaso en la resolución del mismo problema bajo otras condiciones. Se necesita por lo tanto realizar un muy buen análisis de las condiciones del problema que se está enfrentando y tomar decisiones para ese caso particular en función a dichas condiciones.

En esta tesina nos concentramos principalmente en los algoritmos de seguimiento y estimación de trayectorias. Nos interesa particularmente la paralelización de los algoritmos que resuelven el problema, con el fin de obtener un tiempo de respuesta admisible para situaciones que requieren respuestas en tiempo real. A continuación se plantean características y suposiciones que se asumen acerca del problema. Las cuestiones más importantes a tener en cuenta son:

- Origen de los datos, ubicación de la cámara.
- Características de la escena.
- Características de los objetos a seguir y sus movimientos.

En este trabajo se utilizan herramientas automáticas para generar imágenes artificiales que representen movimientos de entidades dentro de una escena ficticia con el fin de realizar pruebas bajo condiciones completamente conocidas, donde se conozca con exactitud el recorrido de cada uno de los objetos. De esta forma se pueden evaluar los resultados obtenidos por el sistema en una forma precisa. Sin embargo, las imágenes utilizadas podrían corresponder a una escena real y ser capturadas en tiempo real para su procesamiento mediante cualquier dispositivo de captura de imágenes.

La utilización de imágenes generadas artificialmente nos permite enfocarnos en el problema que nos concierne, que es el del seguimiento y estimación de trayectorias, dejando de lado los inconvenientes que surgen en la captura de imágenes en tiempo real. Sin embargo, se realiza un breve análisis de las cuestiones que deben tenerse en cuenta en una situación real. Entre ellas se mencionan las diferentes distorsiones que pueden ocurrir en la captura, las cuales obligan a realizar una corrección de las imágenes mediante técnicas de calibración de la cámara y otros algoritmos de procesamiento de imágenes.

Se supone que la escena donde se deslizan los móviles de interés, puede ser capturada completamente por una cámara estática en un plano paralelo a ella a una determinada altura. Esta suposición permite trabajar en un plano de dos dimensiones, en lugar de trabajar en un espacio de tres dimensiones. Se trabaja con la proyección de los objetos sobre el plano por el cual se deslizan. El eje de coordenadas que se utiliza es el del plano donde se proyectan los objetos, que es donde se desplazan.

Con respecto a las características de los móviles, la única condición que se establece es que las figuras correspondientes a los objetos son oscuras y el fondo sobre el cual se desplazan es claro. No se realiza ninguna otra suposición ni se conocen de antemano los objetos a seguir. Por lo tanto, no se dispone de características que permitan identificarlos tales como la forma o el tamaño. Bajo este escenario los objetos pueden ser similares entre sí. Este punto afecta particularmente a las etapas de segmentación y reconocimiento de los objetos y a la etapa de armado de las trayectorias.

En cuanto a las características del movimiento, se supone que los objetos se desplazan en forma lineal a velocidad constante (movimiento rectilíneo uniforme) dentro de la escena, pudiendo desaparecer de ella de forma temporal o definitiva por excederse de los límites captados por el dispositivo de captura. La dirección del movimiento puede variar en un instante determinado de tiempo.

2.3 Como afectan las suposiciones realizadas en la resolución del problema

Trabajar con imágenes generadas en forma automática, además de facilitar la generación de datos de prueba, simplifica los primeros pasos en la resolución del problema, ya que desaparecen los inconvenientes relacionados con la captura de imágenes. La necesidad de aplicar algoritmos de mejora de la imagen depende de las características de las imágenes obtenidas. Los datos de prueba pueden ser creados de tal forma que sea necesaria la aplicación de algoritmos clásicos de procesamiento de imágenes antes de la etapa de segmentación, con el fin de resolver problemas de iluminación, distintos tipos de ruido, deformaciones geométricas, etc. En nuestro caso las imágenes usadas no necesitarán la aplicación de muchos algoritmos de preprocesado. De esta forma se simplifican las primeras etapas.

3 Conceptos básicos

3.1 Visión artificial

3.1.1 Introducción

La visión artificial es una disciplina que tiene como finalidad la extracción de información del mundo real a partir de imágenes, utilizando para ello una computadora. Se trata de un objetivo ambicioso y complejo.

Un sistema de visión artificial actúa sobre una representación de una realidad que le proporciona información sobre brillo, colores, formas, entre otras cosas. Estas representaciones suelen estar en forma de imágenes estáticas, escenas tridimensionales o imágenes en movimiento.

Los primeros sistemas de visión artificial estaban relacionados con imágenes estáticas. Actualmente, existe una gran cantidad de sistemas que trabajan con imágenes dinámicas. La entrada a un sistema de análisis de escenas dinámicas es una secuencia de cuadros o frames tomados de una escena real, donde cada uno de ellos es una representación de la escena observada en un instante preciso de tiempo.

Existen diversos tipos de aplicaciones de visión artificial. Entre otras podemos mencionar el control de calidad de productos en una planta industrial; el diagnóstico de enfermedades o irregularidades utilizando imágenes obtenidas a través de tomografías computadas; el reconocimiento de escenas tridimensionales, útil por ejemplo para la automatización de navegación de autos, aviones y robots; el procesamiento de grandes volúmenes de datos obtenidos por satélites, imágenes que son importantes para diversas aplicaciones, entre las que se pueden mencionar el pronóstico del tiempo, el análisis de la topología del terreno y el comportamiento de la agricultura.

Un campo relacionado con la visión artificial es el procesamiento de imágenes. Por *procesamiento digital de imágenes* se hace referencia generalmente a procesar una imagen bidimensional por medio de una computadora. Las técnicas de procesamiento de imágenes generalmente transforman una imagen en otra. Se incluyen aquí los algoritmos de realce, de corrección de imágenes borrosas o fuera de foco, de compresión, transformaciones geométricas, etc. Por otro lado, los algoritmos de visión por computadora tienen como entrada imágenes, pero la salida es otro tipo de información, como por ejemplo, la representación del contorno de un objeto, las posiciones de los objetos encontrados en la escena, etc. Los algoritmos de procesamiento de imágenes son útiles en las primeras etapas de un sistema de visión artificial.

La Visión Artificial, también llamada Visión Computacional, intenta reproducir el comportamiento del ser humano (Gráfico 3-1).

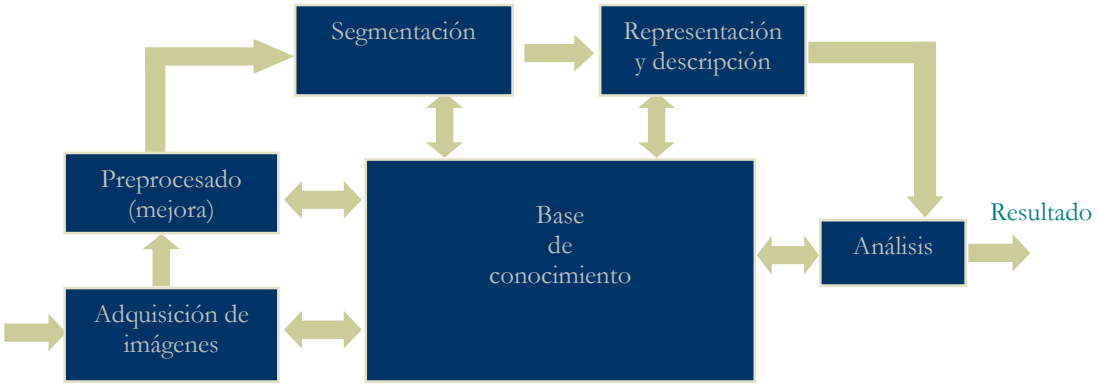


Gráfico 3-1. Visión artificial: Etapas principales del procesamiento.

La primera etapa del proceso es puramente sensorial, consiste en la Captura o Adquisición de las imágenes digitales. Para ello se necesita un sensor y la posibilidad de digitalizar la señal producida por éste. Si la salida del sensor no es digital puede emplearse un conversor analógico-digital para digitalizarla

La segunda etapa consiste en el tratamiento digital de las imágenes, con objeto de obtener una imagen de mejor calidad para facilitar las etapas posteriores. En esta etapa de preprocesamiento es donde mediante la aplicación de filtros y transformaciones se corrigen distorsiones, se eliminan partes indeseables de la imagen o se realzan partes interesantes de la misma.

La segmentación consiste en partir una imagen de entrada en sus partes u objetos constituyentes. En general, la segmentación autónoma es una de las labores más difíciles del tratamiento digital de imágenes.

La segmentación obtiene los datos de los píxeles en bruto que constituyen las regiones de la imagen. La etapa de representación y descripción consiste en convertir los datos de alguna manera adecuada para su procesamiento por computadora. Debe decidirse la representación de los datos, ya sea como un contorno, como una región completa, o bien una combinación de ambas. Luego debe especificarse un método para describir los datos de forma que se resalten los rasgos de interés. La descripción, también llamada criterio de selección, consiste en extraer rasgos con alguna información cuantitativa de interés o que sean fundamentales para diferenciar una clase de objetos de otra.

La última etapa es la de reconocimiento e interpretación. El reconocimiento es el proceso que identifica a un objeto basándose en la información proporcionada por sus

descriptores. La interpretación implica asignar significado a un conjunto de objetos reconocidos.

Se debe tener en cuenta que siempre se tiene algún tipo de información sobre las imágenes que se deben tratar. Estos datos se encuentran reunidos en la llamada base de conocimiento, cuya complejidad y cantidad de información varia según la aplicación. Dicha base no solo interviene en cada una de las etapas a realizar, sino que también se utiliza en la interacción entre estas.

Los algoritmos y técnicas que se aplican en cada una de las etapas varían considerablemente de acuerdo con las características del problema en particular. También debe aclararse que no siempre existen todos los pasos. Por ejemplo, el mejorar una imagen para interpretaciones visuales humanas rara vez pasa más allá de la etapa de preprocesamiento.

Las fases planteadas no se siguen siempre de manera secuencial, sino que en ocasiones deben realimentarse hacia atrás. Así, es normal volver a la etapa de segmentación si falla la de reconocimiento, o la de preproceso, o incluso a la de captura, cuando falla alguna de las siguientes.

3.1.2 Adquisición de datos del mundo real

Este concepto se refiere a la conversión de señales analógicas provenientes de sensores traductores en digitales. Las señales analógicas producidas por los fenómenos físicos como temperaturas, presiones, vibraciones, ruidos, luminosidad o bien señales eléctricas, son captadas a través de un sensor adecuado que convierte o acondiciona la señal analógica a una señal eléctrica. Para que una señal eléctrica pueda ser procesada por una computadora debe ser convertida en una señal digital.

3.1.3 Adquisición de imágenes

La adquisición de imágenes es el primer paso al proceso de análisis de imágenes digitales. Se puede definir al proceso de adquirir y digitalizar como el proceso de obtener señales eléctricas a través de un sensor con un adecuado sistema óptico, y transformarlas en señales digitales.

Durante la adquisición y digitalización de imágenes se producen una serie de distorsiones de la imagen. El análisis de dichas distorsiones constituye un amplísimo campo de estudio, continuamente en expansión. Es muy importante comprender éste problema y cuales son los factores más importantes que lo originan, y no debe dejar de ser tenido en cuenta en el desarrollo de aplicaciones de Visión por computadora. Ignorar las perturbaciones existentes durante la adquisición y digitalización puede conducir al fracaso en la mayoría de los casos.

Daremos una introducción muy general al proceso de adquisición de imágenes digitales, y mencionaremos a grandes rasgos, cuales son los problemas más comunes que existen durante el proceso de adquisición y digitalización.

3.1.4 Distorsión de imágenes

Durante el proceso de la adquisición de imágenes existen diversos factores a tener en cuenta. Aquí se presenta una introducción a los problemas más comunes.

La correcta interpretación de las imágenes está limitada por:

- Distorsión geométrica debido a la posición relativa del sensor y la escena (deformación propia de la perspectiva).
- Distorsión producida por el sistema de iluminación utilizado (sombra o reflexión).
- Pérdida de información debida al hecho de que la imagen obtenida es generalmente una proyección de dos dimensiones de una escena de tres dimensiones.

3.1.4.1 Distorsión geométrica

Puede ser producida por la perspectiva resultante de la posición de la cámara respecto de la escena y los parámetros de la lente usados.

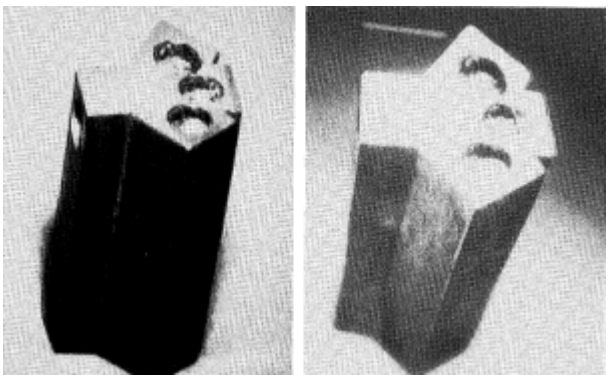


Gráfico 3-2. Distorsión geométrica

Puede confundirse la imagen de la izquierda por un objeto deformado y con una base redondeada, sin embargo ese efecto es producido por una distorsión en la adquisición de la perspectiva. En la imagen de la derecha vemos que el objeto es totalmente rectilíneo y carece de deformaciones (Gráfico 3-2).

3.1.4.2 Distorsión producida por el sistema de iluminación utilizado

Las distorsiones debidas al sistema de iluminación pueden derivar entre otras cosas a alguno de los siguientes problemas:

- Distorsión de la forma y dimensión: Es el resultado de la proyección de la sombra en el plano de la escena. Puede interpretarse la sombra como un área del objeto. Puede conducir a un corrimiento en los límites entre el objeto y el fondo, alterando las magnitudes del objeto observado.

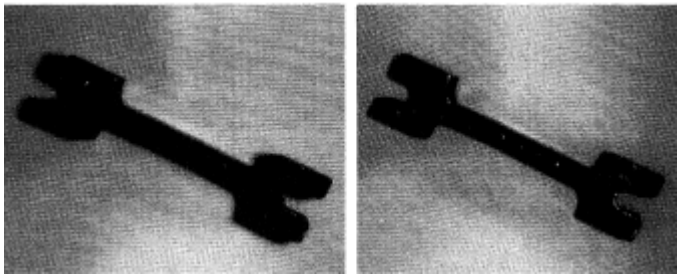


Gráfico 3-3. Distorsión de la dimensión causada por la sombra

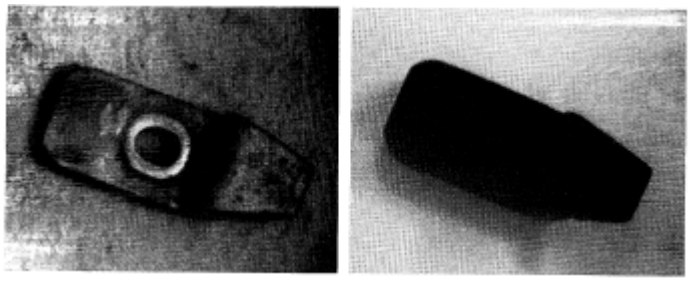


Gráfico 3-4. Oclusión por sombra de características relevantes de un objeto

- Encubrimiento de información: efecto causado por la proyección de la sombra del objeto de la escena. Consiste en el ocultamiento de información relevante a su reconocimiento (surcos, agujeros, uniones, bordes, etc.)

3.1.5 Calibración de la cámara

Se mencionó anteriormente que uno de los problemas principales que ocurren durante la captura de imágenes por medio de una cámara se debe a que la imagen obtenida es una proyección en dos dimensiones de una porción del espacio real en 3 dimensiones.

La calibración consiste en establecer una correspondencia entre los puntos del espacio y de la imagen según una función lo más parecida posible a la transformación de la cámara. Si se representan los puntos del espacio por sus coordenadas (X,Y,Z) en una base ortonormal R_{absolu} de R_3 y los puntos en la imagen por sus coordenadas (u,v) en una base ortonormal del plano, el problema se convierte entonces en hallar una correspondencia entre (X,Y,Z) y (u,v).

El primer paso para calibrar una cámara es *modelar* la transformación del sistema cámara-tarjeta de adquisición (en lo que sigue, "cámara"). Este modelo debe contemplar dos elementos: la física de la cámara y la discretización de la tarjeta.

Definido el modelo, la calibración de la cámara consiste en ajustar al modelo de forma que simule lo "más fielmente posible" el comportamiento de la cámara.

Se denomina *imagen adquirida* de un objeto a la imagen del objeto tomada con la cámara, e *imagen por el modelo* de un objeto a la imagen del objeto a través del modelo de la cámara.

La calibración de la cámara consiste en determinar los parámetros del modelo, haciendo corresponder alguna propiedad o característica de la imagen adquirida de un objeto de calibración (patrón de calibración) a la propiedad correspondiente de la imagen por el modelo del mismo.

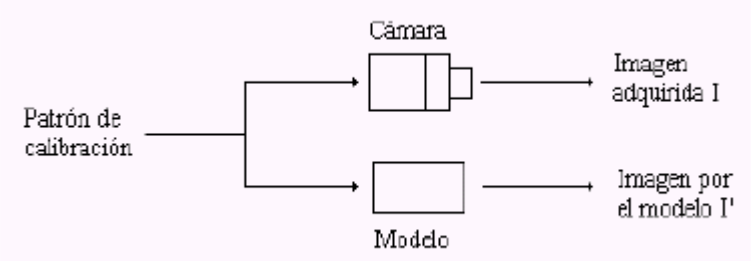


Gráfico 3-5 En la calibración de cámara se comparan características de I con características de I´.

Entre las aplicaciones inmediatas de la calibración monocular (sistema compuesto por una única cámara) se pueden citar las siguientes:

- Medición de distancias en un plano, esto es, conocer distancias reales (absolutas) entre puntos del plano, a partir de sus distancias en píxeles en la imagen.
- Corrección de distorsiones ópticas.
- Reconstrucción de objetos planos a partir de sus imágenes.

En casos muy especiales también es posible determinar la posición de la cámara respecto de un punto de referencia del espacio.

Tanto para medir como para reconstruir objetos, se hace necesario establecer una correspondencia *invertibl*e, y por tanto *biyectiv*a, entre los puntos del espacio y los puntos de la imagen. Para una calibración monocular de cámara, la correspondencia es biyectiva en tanto los puntos del espacio observados estén en un plano.

Si el modelo contiene funciones matemáticas invertibles que describen las distorsiones ópticas, la corrección de las distorsiones se reduce a invertir dichas funciones, una vez ajustadas.

Para la determinación de la posición de la cámara es necesario que el modelo de cámara contemple un desplazamiento rígido respecto de un punto de referencia. La determinación de la posición se reduce entonces al ajuste de los parámetros del modelo que describan dicho desplazamiento rígido.

En resumen, la corrección de distorsiones y la determinación de la situación de la cámara respecto de un punto del espacio surgen de la elección de un modelo adecuado de la cámara.

Las imágenes de la figura (Gráfico 3-6) muestran el resultado de un proceso de calibración. Las imágenes corresponden a una grilla con círculos espaciados uniformemente. Puede observarse a simple viste la distorsión existente en la imagen de la izquierda, la cual está sin calibrar. La imagen de la derecha muestra la misma imagen luego de aplicarse el proceso de calibración, donde la distorsión existente ha sido corregida.

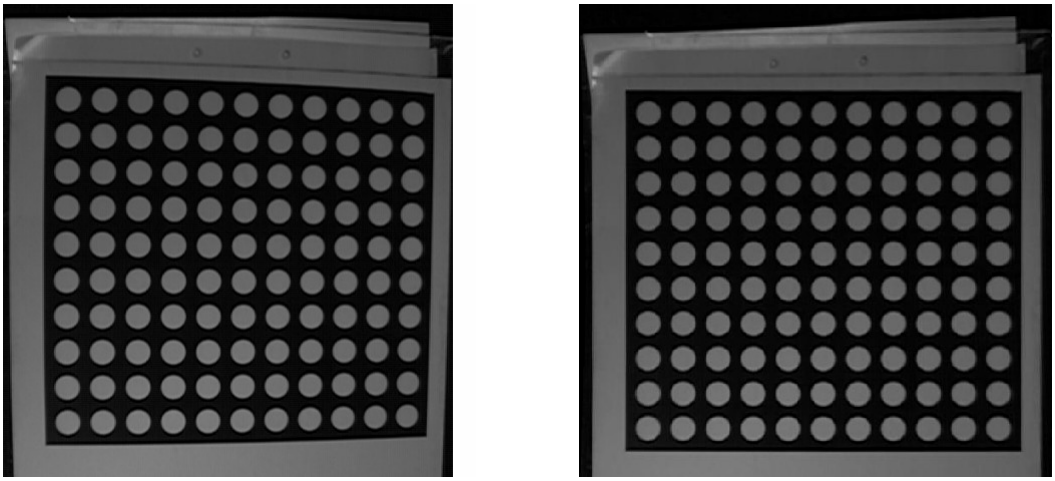


Gráfico 3-6. Calibración de imágenes

3.1.6 Mejora de la imagen

El objetivo principal de las técnicas de mejora es realizar un procesamiento en una imagen de manera tal que resulte más adecuada que la original para una aplicación específica. La palabra *específica* indica que las técnicas de mejorado de la imagen están en gran medida orientadas al problema concreto. Así, por ejemplo, un método que resulte útil para procesar imágenes de rayos X puede no ser necesariamente el ideal para el mejorado de imágenes provenientes de un satélite.

Los algoritmos de mejora de la imagen se aplican en las primeras etapas de un sistema de visión por computadora. En algunas aplicaciones sencillas estos algoritmos son suficientes para resolver el problema en particular. Como ejemplo de ello podemos mencionar una simple mejora del contraste en una imagen tomada por un tomógrafo, con el fin de facilitar la visualización por parte del personal especializado.

Existe un gran número de técnicas de mejora de la imagen, y como es de esperar, sigue siendo un área en continuo desarrollo.

Los métodos de mejora de la imagen pueden agruparse en dos grandes categorías: los que trabajan directamente sobre los píxeles de la imagen; y los que trabajan sobre alguna transformada de la imagen (muchos métodos que se basan en la modificación de la transformada de Fourier de la imagen). Se suele referir a esas dos clases de métodos como procesamiento en el *dominio espacial*, y procesamiento en el *dominio de la frecuencia*, respectivamente. Son habituales otras técnicas que se basan en la utilización de métodos combinados de ambas categorías.

3.1.6.1 Métodos en el dominio espacial

En esta categoría están los procedimientos que actúan directamente sobre los píxeles de una imagen. Las funciones de procesamientos de imágenes en el dominio espacial se pueden expresar como:

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

donde f(x,y) es la imagen de entrada y g(x,y) es la imagen obtenida a partir de la aplicación del operador T, el cual actúa sobre f, definido en algún entorno de (x,y). Para definir un entorno alrededor de (x,y), se suele emplear un área de subimagen cuadrada o rectangular centrada en (x,y). El centro de la subimagen se mueve píxel a píxel aplicando el operador T en cada posición (x,y) para obtener g. Otros tipos de entornos, como aproximaciones a círculos, son más difíciles de implementar, por ese motivo los cuadrados y rectángulos son los más usados.

3.1.6.2 Métodos en el dominio de la frecuencia

Estas técnicas se basan en el teorema de convolución. Sea g(x,y) una imagen formada por la convolución de una imagen f(x,y) y un operador lineal invariante de posición h(x,y) (el resultado depende sólo del valor de f(x,y) en un punto de la imagen, y no de la posición del punto), es decir:

$$g(x,y) = h(x,y) * f(x,y)$$

Por el teorema de convolución, se cumple la siguiente relación en el dominio de la frecuencia:

$$G(u,v) = H(u,v)F(u,v)$$

Donde G, H y F son las transformadas de Fourier de g, h y f, respectivamente.

3.1.7 Segmentación

La segmentación se realiza una vez que una imagen ya ha sido preprocesada y mejorada. Normalmente, los algoritmos aplicados durante la etapa de preprocesamiento tienen como objetivo obtener una nueva imagen a partir de la original que sea más adecuada al momento de realizar la segmentación.

La segmentación consiste en identificar los componentes significativos de la imagen, o sea subdividir la imagen en sus partes constituyentes u objetos. El nivel de detalle al que se llega depende del problema a resolver, lo que significa que la segmentación se da por finalizada cuando los objetos de interés de la aplicación han sido aislados.

Para citar un ejemplo supongamos estar tomando imágenes de una carretera con el objetivo de obtener una estadística de los autos que circulan por ella. El primer paso para ello sería segmentar el camino de la imagen y a continuación segmentar del contenido del camino los objetos cuya longitud o tamaño sean potenciales vehículos. No tiene sentido llevar la segmentación a mas detalle, ni hay necesidad de intentar segmentar componentes que estén fuera del camino.

Los algoritmos de segmentación de imágenes monocromáticas generalmente se basan en una de las propiedades básicas de los valores del nivel de gris: discontinuidad y similaridad.

En la similaridad, los métodos se basan en dividir la imagen en función de la intensidad luminosa de los píxeles; entre los principales métodos de esta categoría pueden nombrarse la umbralización y el etiquetado.

En la discontinuidad el método consiste en dividir una imagen basándose en los cambios bruscos de nivel de gris, las principales áreas de interés de esta categoría son la detección de puntos, líneas y bordes de una imagen.

La mayoría de los métodos de detección de discontinuidades se basan en la aplicación de máscaras de convolución a la imagen. De esta manera sea H una máscara de tamaño p x p y U una imagen arbitraria, la aplicación de la máscara de convolución se obtiene mediante el producto interior de ambos en una localización (m,n), siendo éste equivalente a la siguiente correlación:

$$\sum_i \sum_j h(i,j)u(i+m, j+n)$$

3.1.7.1 Detección de puntos

La idea para detectar un punto aislado es medir la intensidad de gris del punto que se está analizando y compararlo con sus puntos vecinos, ya que si es un punto aislado tendrá un valor bastante diferente del de sus vecinos.

Dada la siguiente máscara, se sabe que se ha detectado un punto en la posición en la que está centrada la mascará si:

$$|R| > T$$

donde **T** es un umbral no negativo y **R** es el valor resultado de aplicar la máscara.

-1	-1	-1
-1	8	-1
-1	-1	-1

3.1.7.2 Detección de líneas

Las siguientes máscaras pueden ser utilizadas para detectar líneas en una imagen.

Analicemos la primera máscara. Si se traslada por toda la imagen responderá con más fuerza a líneas orientadas horizontalmente. Con un fondo constante la respuesta máxima resultará cuando la línea pase por la fila central de la mascara.

De la misma manera resultará la segunda máscara para líneas a 45º, la tercera para líneas verticales y la cuarta para líneas a –45º.

-1	-1	-1
2	2	2
-1	-1	-1

-1	-1	2
-1	2	-1
2	-1	-1

-1	2	-1
-1	2	-1
-1	2	-1

2	-1	-1
-1	2	-1
-1	-1	2

3.1.7.3 Detección de bordes

Si bien los puntos y líneas son elementos de cualquier imagen la detección de bordes es la mas utilizada. La razón es que los puntos aislados y las líneas delgadas no son de frecuente

aparición en la práctica. Se define a un borde como la frontera entre dos regiones con propiedades de nivel de gris relativamente distintas.

Esta técnica es muy usada cuando las regiones en cuestión son lo suficientemente homogéneas para que la transición entre dos de ellas se pueda determinar sobre la base de las discontinuidades de nivel de gris solamente.

La detección de bordes se basa en analizar el gradiente de la imagen (primera derivada discreta). Los bordes de la imagen aparecerán como máximos locales. Existen diversas máscaras que permiten calcular el gradiente. En las siguientes figuras se muestran algunas de ellas. Los filtrados de Prewitt y Sobel emplean máscaras de 3x3 para cada dirección espacial. El uso de máscaras en distintas direcciones permite calcular la dirección del gradiente además de su valor.

Prewitt

1	1	1
0	0	0
-1	-1	-1

-1	0	1
-1	0	1
-1	0	1

Sobel

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

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

La mayoría de las técnicas de detección de bordes realizan los siguientes pasos:

- Calcular una imagen de gradiente (los bordes aparecerán como máximos locales idealmente)
- Umbralizar la imagen de gradiente para obtener sólo los puntos que superan un cierto valor.
- Los puntos con gradiente alto pueden reforzarse en función del valor de sus puntos vecinos, para obtener zonas continuas de contorno.
- Puede haber zonas donde los bordes desaparecen o presentan discontinuidades. Mediante el seguimiento de los mismos se intenta unir las distintas zonas hasta obtener contornos completos.

El algoritmo de Canny está considerado como uno de los mejores métodos de detección de contornos mediante el empleo de máscaras de convolución optimando tres criterios: de detección, localización y respuesta única ante un borde.

El algoritmo consiste básicamente en cinco fases:

- Suavizado de la imagen mediante una máscara Gaussiana para reducir los efectos del ruido.
- Convolución de la imagen por operadores de gradiente direccionales.
- Supresión de los gradientes que no son máximos en la dirección del contorno.
- Aplicación de un umbral adaptativo con histérisis para distinguir los contornos reales del ruido.
- Integración de los contornos detectados a diferentes escalas en un solo mapa de contornos consistentes.

3.1.7.4 Umbralización

La umbralización es una técnica sencilla que resulta útil cuando los píxeles que conforman los distintos objetos se caracterizan por tener una intensidad de nivel de gris bien diferenciada.

El histograma de niveles de gris h(z) de una imagen representa la frecuencia del valor de gris z en la imagen. Supongamos que el histograma de la figura (Gráfico 3-6) corresponde a una imagen f(x,y), compuesta por objetos luminosos sobre un fondo oscuro, de tal forma que los píxeles del objeto y del fondo tienen niveles de gris agrupados en dos modos dominantes.

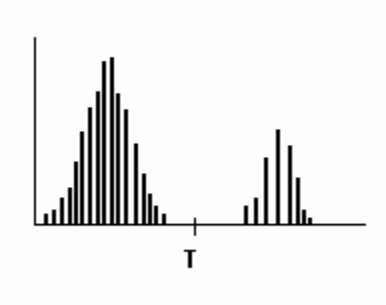


Gráfico 3-7. Histograma 1

Una forma evidente de extraer los objetos del fondo es elegir un umbral T que separe de dichos nodos. Entonces cualquier punto (x,y) para el que f(x,y)>T se denomina punto del objeto; en caso contrario se denomina punto del fondo.

Analicemos un caso mas general de esta aproximación (Gráfico 3-8). Aquí, los tres modos dominantes caracterizan el histograma de la imagen (por ejemplo, dos tipos de objetos claros sobre fondo oscuro). La misma aproximación básica clasifica un punto (x,y) como

integrante de la clase de uno de los objetos si T1<f(x,y)<T2, a la otra clase de objetos si f(x,y) >T2, y al fondo si f(x,y)<T1. Este tipo de umbralización multinivel suele ser menos segura que su contrapartida del umbral único. La razón es la dificultad de establecer umbrales múltiples que aíslen efectivamente las regiones de interés, sobre todo cuando el número de modos de los histogramas correspondientes es grande. Normalmente, los problemas de esta naturaleza, si se tratan por umbralización, se estudiaran mejor por un umbral único que por uno variable.

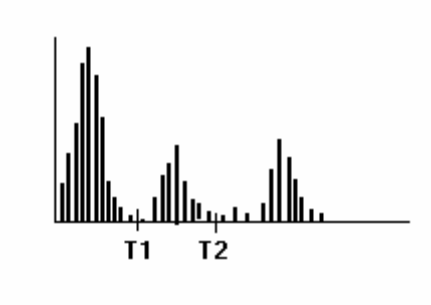


Gráfico 3-8. Histograma 2

Basándose en lo expuesto anteriormente, la umbralización se puede contemplar como una operación que implica realizar comprobaciones frente a una función T de la forma:

T = T[x, y , p(x,y), f(x,y)]

Donde f(x,y) es el nivel de gris del punto (x,y), y p(x,y) representa alguna propiedad local de este punto, por ejemplo, la media el nivel de gris de una vecindad centrada en (x,y). Una imagen umbralizada se define como:

$$G(g,y) = \begin{cases} 1 \text{ si } f(x,y) > T \\ 0 \text{ si } f(x,y) <= T \end{cases}$$

De este modo los píxeles marcados con 1 (o con cualquier otro nivel de intensidad conveniente) corresponde a objetos, mientras que los píxeles marcados con 0 corresponden al fondo.

Cuando T depende solamente de f(x,y), el umbral se denomina global, si T depende a la vez de f(x,y) y p(x,y), el umbral se denomina local. Si además T depende de las coordenadas espaciales x e y, el umbral se denomina dinámico. Cuando el umbral es global, es decir, es constante para todos los píxeles de la imagen, hay situaciones donde su valor puede ser

establecido de antemano. En caso contrario, existen diversas técnicas que determinan el valor óptimo para el umbral analizando el histograma de la imagen.

3.1.7.5 Etiquetado

El etiquetado es una técnica de segmentación de imágenes bastante simple y efectivo para determinadas imágenes. Consiste en analizar la conectividad de píxeles vecinos y marcar con una misma etiqueta al conjunto de píxeles similares y conexos. Como resultado del proceso de etiquetado cada una de las regiones existentes en la imagen queda marcada con una etiqueta distinta.

A continuación se presenta un pseudocódigo del algoritmo.

```
Inicializar todos los píxeles etiquetados con 0.
Inicializar etiqueta_actual a 1

Inicializar el conjunto de píxeles ligados en vacío (p_ligados)

Para cada píxel p en la imagen
      Si(p esta etiquetado con 0) y (vecinos "similares" de p > T)
      Insertar p en p_ligados
      Mientras (p_ligados no este vacío)
            Sacar un píxel b de p_ligados
            Etiquetar a b con etiqueta_actual
            Para cada a en la vecindad de b
                  Si (a no esta etiquetado) y (a es "similar" a b) y (a
                  no esta en p_ligados)
                        Insertar a en p_ligados
      Incrementar etiqueta_actual
```

El algoritmo recorre la imagen buscando nuevas regiones. Se considera que un píxel no etiquetado conforma una nueva región si tiene suficientes (al menos T) vecinos similares a él. El concepto de "similares" determina si dos píxeles son lo suficientemente parecidos como para poder pertenecer a la misma región, lo cual puede ser tan sencillo como un umbral (una tolerancia) entre la diferencia de intensidad de los dos puntos. Esto significa que un píxel con intensidad p es considerado similar a píxeles con intensidades en el rango [p – r, p + r]; por lo tanto valores chicos de r obtendrán regiones suaves mientras que valores grandes podrán mezclar regiones. La definición de vecindad de un píxel afecta el tamaño y número de regiones y el valor de T determina su tamaño mínimo.

Una vez que se determina que un píxel conforma una nueva región comienza su expansión. La región se extiende a través de los píxeles vecinos similares sucesivamente hasta llegar a los límites de ella donde los vecinos dejan de ser "similares". Cuando la expansión

termina todos los píxeles que conforman la región quedan etiquetados con el mismo valor. Luego se continúa recorriendo la imagen buscando un nuevo píxel que determine otra región.

3.2 Procesamiento paralelo

Históricamente el procesamiento paralelo ha sido la única forma de afrontar algunos problemas de procesamiento. Existen muchas aplicaciones en la cuales es necesario procesar grandes cantidades de datos en muy poco tiempo. En situaciones donde la velocidad es crucial, como es el caso de los sistemas que requieren respuestas en tiempo real, y tales respuestas no pueden ser alcanzables mediante soluciones secuenciales con un único procesador, el procesamiento paralelo se presenta como una buena alternativa, sino la única, para afrontar el problema.

3.2.1 Clasificación de las arquitecturas de procesamiento

Se han propuesto muchas formas de organizar las arquitecturas de procesamiento paralelo. La clasificación más popular es la de Flynn y se centra en la forma en que se ejecutan las instrucciones sobre los datos. Cualquier computadora, sea paralela o no, opera ejecutando instrucciones sobre los datos. Flynn distingue cuatro clases básicas:

- **SISD** (Single Instruction stream, Single Data stream): las computadoras de este tipo son las de procesamiento secuencial clásico, donde las instrucciones se ejecutan en serie, una detrás de la otra. Se pueden encontrar en estas computadoras una unidad de procesamiento encargada de llevar a cabo la ejecución de las instrucciones, una unidad de control encargada de decodificar las instrucciones y enviar las señales de ejecución correspondientes a la unidad de procesamiento y la unidad de memoria en donde se almacenan tanto datos como instrucciones.
- **MISD** (Multiple Instruction stream, Single Data stream): en estas computadoras un mismo dato es procesado por múltiples instrucciones en distintas unidades de procesamiento.
- **SIMD** (Single Instruction stream, Multiple Data stream): las computadoras con esta arquitectura constan de un conjunto de elementos de procesamiento idénticos, todos controlados por una única unidad de control. La ejecución es sincrónica, dado que hay una unidad de control compartida y un reloj global. Cada instrucción que se ejecuta en un elemento de procesamiento procesa datos distintos de los demás.

- **MIMD** (Multiple Instruction stream, Multiple Data stream): esta clase de computadoras es la más flexible y genérica. Las computadoras que se incluyen dentro de esta clase constan de *n* procesadores, donde cada uno puede ejecutar su propia secuencia de instrucciones. Cada secuencia de instrucciones se ejecuta en un procesador con diferentes datos de los que se utilizan en los demás procesadores.

Dentro de la clase MIMD se pueden diferenciar dos subclases en función de la forma en que se conectan los procesadores a la memoria y entre sí, los multiprocesadores y las multicomputadoras.

Los multiprocesadores, o computadoras fuertemente acopladas son máquinas con arquitectura MIMD donde todos los procesadores comparten una memoria única, o sea el espacio de direccionamiento es común a todos ellos.

En las multicomputadoras o computadoras débilmente acopladas cada procesador tiene su propia memoria. A través de una red de interconexión un procesador puede enviar mensajes a los demás procesadores.

3.2.2 Paradigmas de programación

Las características de la arquitectura tienen influencia en el paradigma de programación utilizado en un sistema paralelo. La programación paralela puede dividirse en dos amplias categorías: sistemas acoplados en forma débil o fuerte.

En los sistemas fuertemente acoplados todos los procesos tienen acceso a una única memoria compartida. En este caso los procesos se comunican escribiendo y leyendo datos de la memoria. Debe disponerse de mecanismos que sincronicen el uso compartido de la memoria, de modo que solo un proceso escriba una variable compartida a la vez, y que un proceso pueda esperar hasta que ciertas condiciones se satisfagan por completo antes de continuar la ejecución.

En los sistemas débilmente acoplados cada proceso tiene un espacio exclusivo de direcciones de memoria y la comunicación se realiza a través de mecanismos de pasaje de mensajes entre procesos.

3.2.3 Clusters de Pc's

Los clusters de PC's entran en la clase de arquitectura paralela MIMD (Multiple Instruction stream, Multiple Data stream). Es una arquitectura de memoria distribuida y está formada por múltiples nodos interconectados.

Los nodos que componen los clusters son PCs de escritorio y la red de interconexión es una red Ethernet, la cual utiliza el protocolo 802.3 (CSMA/CD). Las velocidades de

transferencia usualmente son superiores a 100 Mb/s. El protocolo 802.3 conecta todas las computadoras a través de un único medio físico compartido. Los mensajes se transmiten desde un emisor y ocupan el bus compartido, por lo tanto, llegan a todas las computadoras que componen el cluster.

Si todas las computadoras que componen el cluster tienen la misma capacidad (de procesamiento, almacenamiento, etc) el cluster es homogéneo. Si por el contrario, tienen distintas capacidades, el cluster es heterogéneo.

En síntesis, se utiliza una red local (red LAN) como computadora paralela.

3.2.4 Speedup (Aceleración)

La medición de rendimiento en ambientes paralelos sirve para determinar cuál es el beneficio obtenido al usar paralelismo y cuál es la aceleración que resulta por el uso de dicho paralelismo. La métrica mas utilizada para determinar el beneficio de un sistema paralelo es el speedup, que se define como el cociente entre el tiempo de ejecución secuencial y el tiempo de ejecución paralelo.

$$\text{Speedup} = \frac{Ts}{Tp}$$

Donde Ts es el tiempo de ejecución secuencial y Tp el tiempo de ejecución utilizando una solución paralela.

4 Problema analizado

4.1 Introducción

El problema que se resuelve en esta tesis es el de seguimiento y estimación de las trayectorias realizadas por un conjunto de objetos en movimiento dentro de una escena. La escena de interés puede ser capturada por una cámara ubicada a una cierta altura en un plano paralelo al plano sobre el cual se desplazan los objetos. Si se proyecta la posición de la cámara sobre el plano de la escena, la cámara queda ubicada en el centro de ella. De esta forma, en lugar de trabajar en un espacio de tres dimensiones, se trabaja en un plano de dos dimensiones, con la proyección de los objetos sobre el plano sobre el cual se deslizan.

Este tipo de situaciones es común, por ejemplo, en las competencias de fútbol de robots. El campo de juego se puede cubrir fácilmente mediante una cámara colgada del techo, pudiendo disponer de una visión global del escenario, simplificando de esta manera la detección de la posición de los robots.

A continuación se explican las etapas involucradas en la resolución del problema (Gráfico 4-1).

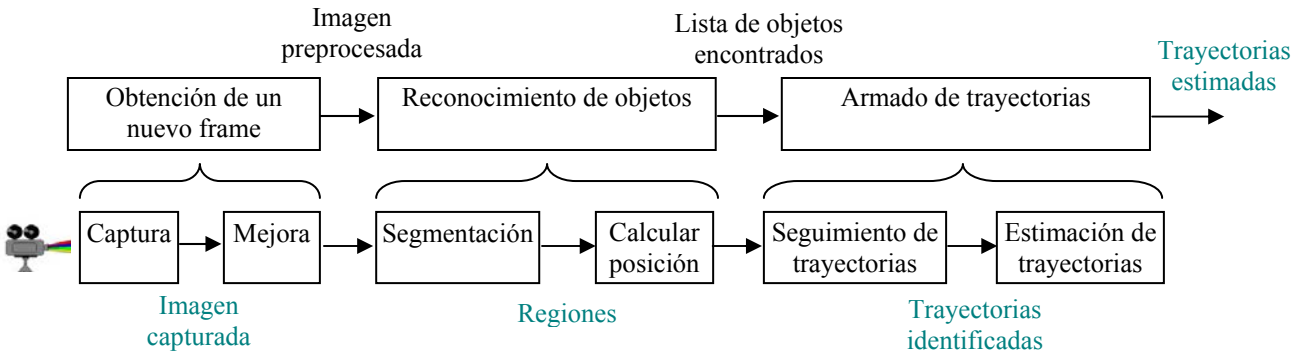


Gráfico 4-1. Ciclo de resolución del problema

A grandes rasgos podemos dividir el sistema en tres grandes etapas, la primera consiste en la obtención de un frame para el procesamiento, abarcando tanto la captura como un preprocesamiento de la imagen con el objetivo de mejorar su calidad. La segunda etapa consiste en el reconocimiento de los objetos de la escena, para lo que es necesario realizar primero una segmentación de la imagen para aislar las regiones que constituyen los objetos de interés y luego deben calcularse las coordenadas de cada uno de ellos en la escena. En la tercer etapa se arman las trayectorias; primero se identifica el recorrido realizado por cada uno de los

objetos a lo largo de la secuencia de frames analizada y en función a dicha información se realiza luego una estimación de las trayectorias futuras.

Con el objetivo de enfocarnos en la resolución del problema de seguimiento y estimación de las trayectorias y en la paralelización de los algoritmos con el objetivo de mejorar los tiempos de respuesta, las pruebas no se realizan con una filmación de una escena real. En lugar de eso, se utilizan imágenes generadas por herramientas automáticas. Esto permite simular diversos tipos de situaciones y realizar una mayor cantidad de pruebas que se pueden evaluar con una mayor exactitud, ya que en cada caso los movimientos de los objetos son perfectamente conocidos. De esta forma se puede analizar el funcionamiento de los algoritmos de seguimiento y predicción de una forma más precisa. De todas formas se simula el comportamiento que tendría el sistema si tomara datos a través de un dispositivo de captura.

4.2 Obtención de un nuevo frame para el procesamiento

4.2.1 Adquisición del frame

El sistema se encuentra permanentemente en el ciclo mostrado anteriormente. El ciclo comienza con la adquisición de un nuevo frame. En una situación real, en la cual las imágenes son capturadas por un sensor, la adquisición consiste en la recuperación de la información capturada. Esta tarea dependerá de las características del hardware que se disponga. El sistema puede recibir alguna señal de interrupción desde el sensor cuando éste disponga de nuevos datos; puede chequear la existencia de un nuevo frame cada un intervalo de tiempo determinado; etc.

En nuestro caso, el sistema es probado con videos generados de antemano en forma automática que son almacenados en archivos. La captura de un nuevo frame consiste simplemente en leerlo desde el archivo de video. Dicha lectura debe respetar, sin embargo, la velocidad original del video. Es decir, si el video fue generado usando 20 frames por segundo, es de esperar que se lea un nuevo frame cada 1/20 segundos. Esto se simula mediante el uso un timer, que indica el intervalo de tiempo tras el cual debe realizarse una nueva lectura. De esta forma, el sistema puede adaptarse fácilmente a una situación real modificando el módulo de captura, para que tome los datos desde un sensor en lugar de hacerlo desde un archivo.

El formato de video que utilizamos en nuestro trabajo es mpeg. Para recuperar los frames desde un archivo mpeg se utiliza una librería que es detallada en los apéndices. La librería tiene funciones para leer un frame determinado del archivo y llevarlo a memoria en una estructura matricial.

4.2.2 Calibración

Previamente hemos analizado los problemas que aparecen cuando se capturan imágenes de la realidad mediante un sensor. Hemos visto que cuando se trabaja con un

dispositivo de captura, existen distorsiones geométricas en las imágenes capturadas. Uno de los problemas principales que ocurre durante la captura de imágenes por medio de una cámara se debe a que la imagen obtenida es una proyección en dos dimensiones de una porción del espacio real de 3 dimensiones. La calibración es un método que ya fue presentado y consiste en establecer una correspondencia entre los puntos del espacio y de la imagen según una función lo más parecida posible a la transformación de la cámara. El objetivo es reducir las deformaciones geométricas existentes.

A continuación se muestra un ejemplo del resultado del proceso de calibración (Gráfico 4-2, Gráfico 4-3)

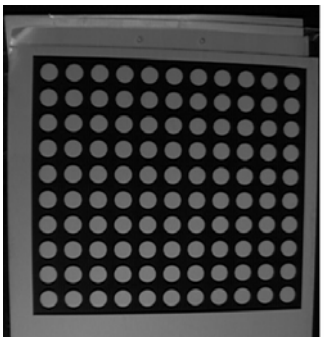


Gráfico 4-2. Imagen sin calibrar

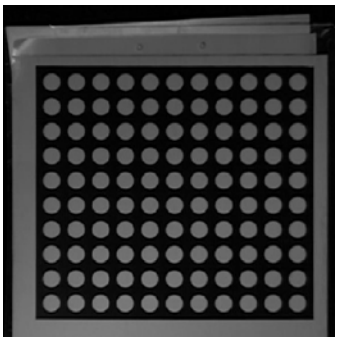


Gráfico 4-3. Imagen calibrada

En nuestro trabajo no aplicaremos los algoritmos de calibración, ya que los videos que utilizamos de prueba no la necesitan.

4.2.3 Mejora de la imagen

Prácticamente en cualquier sistema de visión por computadora, al disponer de una imagen para el procesamiento es necesario aplicarle previamente una serie de algoritmos de mejora. La finalidad de éstos algoritmos fue explicada anteriormente. Existe una gran variedad de algoritmos de mejora y cada uno pretende obtener una imagen que sea más adecuada que la original para el posterior procesamiento.

Debido a que lo que nos interesa es enfocarnos en los algoritmos de seguimiento y estimación de trayectorias, la única característica indeseable que introducimos en los videos generados para probar el sistema es la presencia de ruido impulsivo (sal y pimienta). Esto puede parecer innecesario a los fines de este trabajo, pero el objetivo es que a pesar de que trabajemos con imágenes creadas artificialmente, en el modelo de solución planteada estén todos los pasos que existen en la solución de un caso real. Para eliminar el ruido impulsivo, se realiza un filtro por la mediana.

Nuevamente es necesario destacar que los algoritmos que se apliquen en éste paso dependen de las características de la imagen original. Cuando se trabaja con imágenes

capturadas de la realidad, éstas deben analizarse para determinar las cualidades indeseables que pueden interferir en el posterior procesamiento. Una vez identificadas, se debe decidir cuáles algoritmos de mejora son los adecuados para modificar la imagen de tal forma que sea adecuada para el posterior procesamiento.

4.3 Reconocimiento de los objetos de interés

4.3.1 Introducción

Una vez que se ha adquirido un nuevo frame, y se le han aplicado los algoritmos de mejora necesarios, llega el momento de identificar a los objetos existentes en la imagen. A esta etapa se la conoce como segmentación en el campo de la Visión por computadora.

Existen muchas técnicas de segmentación, cada una de las cuales resulta más o menos adecuada según las características de la imagen a procesar.

En nuestro caso, tenemos una imagen en tonos de grises que corresponde a una escena con fondo claro, donde los objetos que deseamos identificar son oscuros. Es importante mencionar que esas son las únicas condiciones tenidas en cuenta para la segmentación. No se conocen de antemano características de los objetos buscados tales como tamaño o forma. Definimos a un objeto como un conjunto de píxeles conexos, con una intensidad similar y suficientemente diferente a la intensidad del fondo.

Para resolver el problema de la identificación de los objetos de la escena usamos un algoritmo de etiquetado. La técnica de etiquetado consiste en explorar la imagen e ir asignando una etiqueta a cada píxel, de tal forma que los píxeles que corresponden al mismo objeto tengan la misma etiqueta. Se analiza la similitud de un píxel con sus vecinos para determinar si corresponden al mismo objeto o no.

Debido a las características planteadas del problema, antes de implementar el algoritmo de etiquetado, transformaremos la imagen en escala de grises en una imagen bivaluada (binaria), es decir, una imagen con dos intensidades. Una de las intensidades representará los píxeles que corresponden al fondo de la escena, y la otra aquellos píxeles que conforman algún objeto. Esta técnica se llama umbralización, y su aplicación facilita el posterior algoritmo de etiquetado.

4.3.2 Umbralización

La técnica de umbralización ya fue presentada en la sección de Segmentación de Imágenes. Es posible utilizarla en situaciones como la nuestra, donde los píxeles que conforman los objetos y el fondo tienen niveles de gris agrupados en dos modos dominantes. Consiste en elegir un umbral T de manera que los píxeles que corresponden al fondo tengan una intensidad mayor que T, y aquellos que correspondan a un objeto tengan una intensidad menor que T.

Siendo f(x,y) la imagen a procesar, definimos de la siguiente manera a la nueva imagen G(x,y) obtenida mediante la umbralización,

$$G(x,y) = \begin{cases} 1 & \text{si } f(x,y) < T \\ 0 & \text{si } f(x,y) >= T \end{cases}$$

De esta manera los píxeles con valor 1 en G, corresponden a un objeto, mientras que los píxeles con valor 0 corresponden al fondo

El valor del umbral es un parámetro muy importante y debe ser elegido de tal manera que separe efectivamente los objetos del fondo. En nuestro caso, donde conocemos de antemano la intensidad promedio de los píxeles del fondo y de los objetos, el valor del umbral es elegido en forma manual de tal manera que separe las dos clases de píxeles.

El algoritmo para obtener una imagen bivaluada a partir de la original es el siguiente.

```
ENTRADA: imagen en tonos de grises f(x,y), umbral T
SALIDA: imagen binaria G(x,y)

para x desde 0 hasta ancho(f) – 1
      para y desde 0 hasta alto(f) – 1
            si (f(x,y) < T)
                  G(x,y) = 1
            sino
                  G(x,y) = 0
```

4.3.3 Etiquetado

La técnica de etiquetado para realizar la segmentación de una imagen ya fue explicada previamente y se mostró el pseudo-código de un algoritmo para aplicar en el caso general de tener una imagen en tonos de grises. A continuación se muestra un algoritmo de etiquetado que funciona correctamente para imágenes binarias.

```
ENTRADA: imagen f binaria (0-fondo, 1-objeto)
SALIDA: imagen etiquetada E(x,y)

∀x ∀y E(x,y)=0 // inicializa las etiquetas de todos los pixeles en 0
etiqueta_actual = 1
p_ligados = ϕ

para x desde 0 hasta ancho(f) – 1
      para y desde 0 hasta alto(f) – 1
            Si (E[x,y] = 0 y  f(x,y) = 1 )
            // El píxel pertenece a un objeto y no esta etiquetado
```

```
                p_ligados = p_ligados ∪ {(x,y)}
                mientras (p_ligados <> ϕ)
                      Sacar un píxel b=(x',y') de p_ligados
                      E[x',y'] = etiqueta_actual
                      //Etiqueta a b con etiqueta_actual
                      para cada a= (x'',y'') en la vecindad de b
                            Si (E[x'',y''] = 0) y (f[x'',y''] = 1) y
                                  (a ∉ p_ligados)
                                  a es vecino no etiquetado de b
                                  p_ligados= p_ligados ∪ {(x'',y'')}

          etiqueta_actual = etiqueta_actual + 1
```

La salida del algoritmo de etiquetado es una imagen de igual tamaño que la original donde todos los píxeles que corresponden a un mismo objeto tienen el mismo valor (etiqueta).

Pueden eliminarse aquellos objetos cuya superficie (cantidad de píxeles que lo conforman) sea menor a un límite establecido, con el objetivo de no considerar como objetos a partes de la imagen afectadas por ruido.

4.3.4 Determinación de la posición de los objetos de la escena

Una vez que se reconocieron los objetos de interés en un frame, el paso siguiente es determinar sus coordenadas dentro de la escena. Para ello es necesario establecer un sistema de coordenadas. Dado que las imágenes representan una escena capturada desde un plano paralelo superior, se puede trabajar con la proyección de los objetos sobre el plano en el cual se desplazan. Dicho plano representa un sistema de coordenadas de dos dimensiones, y la ubicación de la proyección de un objeto en el plano se puede utilizar para determinar las coordenadas del objeto en la escena.

Existen ciertos factores que deben ser tenidos en cuenta al trabajar de esta forma. A pesar que las imágenes que se usan de prueba en este trabajo se generan en forma automática con el objetivo de poder analizar las respuestas del sistema en escenarios completamente conocidos, hemos analizado de una forma general los problemas que surgen al trabajar con imágenes capturadas de una escena real. Entre ellos podemos mencionar las deformaciones geométricas causadas por la cámara debido a la perspectiva; los problemas originados por trabajar con imágenes de 2 dimensiones que en realidad corresponden a una escena de tres dimensiones (una imagen que corresponde a una escena real es una proyección de 3D en 2D); los problemas de iluminación, etc. Todos estos problemas pueden ser atenuados en las etapas de pre-procesamiento, aunque difícilmente sean eliminados. Si las imágenes corresponden a una escena real filmada de la forma ya especificada, el problema de la deformación debido a la perspectiva puede tener una influencia importante en las etapas de segmentación y cálculo de la

posición de los objetos. En tal caso es necesario que en las etapas de preprocesado deba realizarse alguna calibración de las imágenes para atenuar las deformaciones geométricas.

Un análisis más detallado de todas las cuestiones mencionadas en el párrafo anterior se puede encontrar en este trabajo en el capítulo previo donde se analiza el problema de la captura de datos del mundo real.

Para la determinación de la posición de un objeto dentro de la escena se tendrán en cuenta las siguientes consideraciones:

- En una etapa previa a la etapa de segmentación, la imagen debe haber sido calibrada y mejorada de tal manera de atenuar de la mejor manera posible el problema de la deformación geométrica por la perspectiva, el cual puede derivar en errores graves tanto en la detección y reconocimiento de un objeto como en el cálculo de su posición.
- La secuencia de frames corresponde a una escena en donde la cámara se encuentra en una posición fija. Esto permite establecer un sistema de coordenadas de referencia para poder determinar la posición de cada objeto encontrado en la imagen.
- La posición de los objetos de la escena se calculará en función del área que ocupa en la imagen. Consideramos que un objeto es el conjunto de píxeles relacionados detectados en la etapa de segmentación de la imagen.

El siguiente gráfico (Gráfico 4-4) muestra el sistema de coordenadas que se utilizará para imágenes de tamaño *alto* x *ancho*.

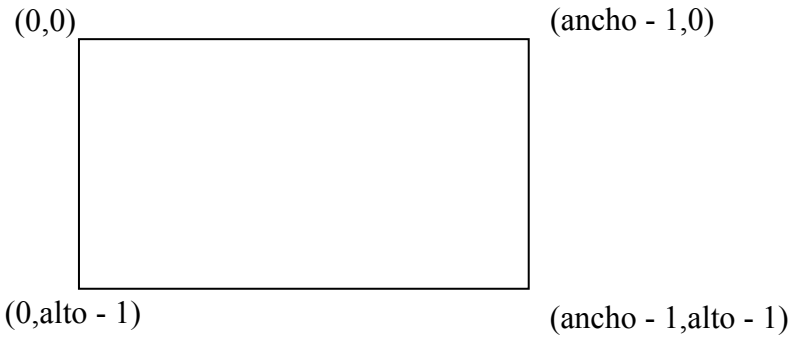


Gráfico 4-4. Sistema de coordenadas

Dado que las imágenes son una proyección de los objetos sobre el plano sobre el cual se desplazan, la ubicación del objeto en la escena está directamente relacionada con las coordenadas de los píxeles que lo constituyen.

4.3.4.1 Problema a resolver

La salida del proceso de segmentación es una serie de regiones, conformada cada una por un conjunto de píxeles, que corresponden a los objetos encontrados en la escena.

El problema a resolver entonces es el siguiente:

Dado un conjunto de píxeles, que representa un objeto en una imagen, determinar las coordenadas que representen la posición del objeto en la escena.

Se utilizará como coordenadas del objeto al centro de masa de la figura que representa al objeto.

4.3.4.1.1 Centro de masa

El centro de masa de un objeto permite localizar un punto singular en el espacio donde pareciera que la masa del objeto se concentrara. Este punto es muy importante en el diseño y construcción de innumerables objetos que usamos a diario. Por ejemplo, los diseñadores de autos siempre tratan de acomodar el centro de masa de un auto lo mas bajo posible, evitando así posibles vuelcos.

El concepto de centro de masa es muy utilizado en la física, ya que de alguna forma representa el movimiento de todo el cuerpo, es decir, el cuerpo se mueve de la misma manera que se movería una sola partícula sometida a las mismas fuerzas externas.

Veremos ahora como se calculan las coordenadas del centro de masa de un cuerpo.

Supóngase un sistema formado por n partículas con coordenadas (x_i, y_i, z_i) y masas m_i en que i = 1,2,.....,n.

El centro de masa de este sistema tendrá coordenadas (x_{cm},y_{cm},z_{cm}) definidas por:

$$x_{cm} = \frac{\sum_{i=1}^{n} m_i x_i}{\sum_{i=1}^{n} m_i} \qquad y_{cm} = \frac{\sum_{i=1}^{n} m_i y_i}{\sum_{i=1}^{n} m_i} \qquad z_{cm} = \frac{\sum_{i=1}^{n} m_i z_i}{\sum_{i=1}^{n} m_i}$$

En donde $\sum_{i=1}^{n} m_i$ = M = masa total del sistema

Un cuerpo rígido, puede considerarse como un sistema de partículas en que la distancia relativa entre ellas permanece constante.

Dado el gran número de partículas y sus espaciamientos tan pequeños, un cuerpo puede considerarse como una gran distribución continua de masa.

Para encontrar el centro de masa, se consideran elementos infinitesimales de masa dm con coordenadas (x,y,z).

En este caso la sumatoria para encontrar las coordenadas del centro de masa tiene un número de sumandos n que tiende al infinito y las coordenadas del centro de masa quedan determinadas por las siguientes integrales:

$$x_{cm} = \frac{1}{m}\int x dm \qquad y_{cm} = \frac{1}{m}\int y dm \qquad z_{cm} = \frac{1}{m}\int z dm$$

En nuestro caso necesitamos calcular el centro de masa de objetos en un espacio de dos dimensiones. Debido a que las figuras que representan los objetos son extraídas de una imagen digital, vamos a considerar a los píxeles como las partículas que conforman a los objetos. Cada píxel tiene masa m_i y coordenadas (x_i,y_i). De esta forma no es necesario utilizar una integral para calcular el centro de masa sino que las siguientes fórmulas nos darán las coordenadas del mismo.

$$x_{cm} = \frac{\sum_{i=1}^{n} m_i x_i}{\sum_{i=1}^{n} m_i} \qquad y_{cm} = \frac{\sum_{i=1}^{n} m_i y_i}{\sum_{i=1}^{n} m_i}$$

Se demuestra fácilmente que la posición del centro de masa de un objeto o sistema de cuerpos es independiente del origen al que estén referidos xcm e ycm, así como a la orientación de los ejes X e Y. Esto es importante ya que el centro de masa es relativo al objeto,

independientemente de su ubicación y orientación. Por ejemplo, el centro de masa de una figura circular siempre será su centro.

4.3.4.2 Cálculo del centro de masa

Ya estudiado el concepto de centro de masa de un objeto, ahora nos concentraremos en el algoritmo para calcularlo. La entrada a dicho algoritmo es el subconjunto de píxeles de la imagen que constituyen al objeto en cuestión, obtenidos en la etapa previa de segmentación. De cada píxel se conocen sus coordenadas dentro de la imagen. La salida del algoritmo son las coordenadas del centro de masa del objeto.

Vamos a considerar a los píxeles como las partículas constituyentes del objeto. Los conceptos dados previamente establecen que las coordenadas del centro de masa de un objeto se calculan en función de las coordenadas y las masas de las partículas que lo constituyen, es decir de los píxeles.

Sea n la cantidad de píxeles que conforman el objeto, m_i la masa del píxel i y (x_i , y_i) sus coordenadas. Consideremos que todos los píxeles tienen masa m.

Las coordenadas del centro de masa (x_{cm}, y_{cm})se calculan con las siguientes fórmulas:

$$x_{cm} = \frac{\sum_{i=1}^{n} m_i x_i}{\sum_{i=1}^{n} m_i} = \frac{\sum_{i=1}^{n} m.x_i}{\sum_{i=1}^{n} m} = \frac{m\sum_{i=1}^{n} x_i}{n.m} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$y_{cm} = \frac{\sum_{i=1}^{n} m_i y_i}{\sum_{i=1}^{n} m_i} = \frac{\sum_{i=1}^{n} m.y_i}{\sum_{i=1}^{n} m} = \frac{m\sum_{i=1}^{n} y_i}{n.m} = \frac{\sum_{i=1}^{n} y_i}{n}$$

Vemos que al suponer masa uniforme en el objeto, las coordenadas resultantes como el centro de masa de un objeto resultan ser un promedio de las coordenadas de los píxeles que lo constituyen.

El algoritmo que calcula el centro de masa es entonces:

```
ENTRADA: el conjunto de píxeles Ps que conforman el objeto del cual se
calcula el centro de masa

SALIDA: las coordenadas del centro de masa xcm, y ycm.

xcm = 0
ycm = 0
masaTotal = 0
```

```
para cada p en Ps hacer
      masaTotal = masaTotal + 1
      xcm = xcm + p.x
      ycm = ycm + p.y

xcm = xcm / masaTotal
ycm = ycm / masaTotal
```

A continuación se muestran algunas figuras que ejemplifican el resultado del algoritmo que calcula del centro de masa. En cada caso, la figura de la izquierda representa el objeto original, y la figura de la derecha muestra la misma figura en donde se marca el centro de masa calculado.

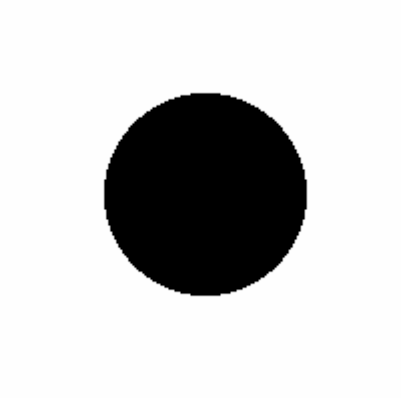


Gráfico 4-5. Figura circular

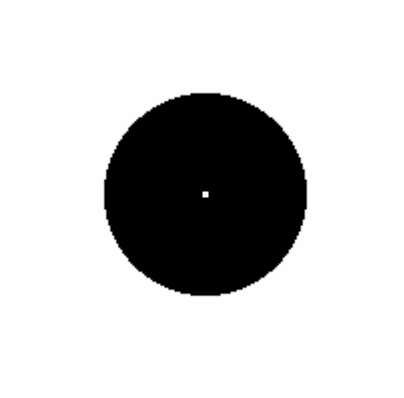


Gráfico 4-6. Cálculo del centro de masa

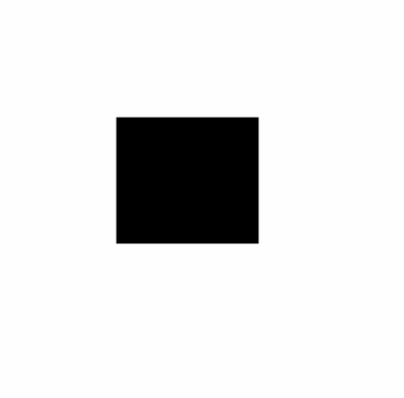


Gráfico 4-7. Figura rectangular

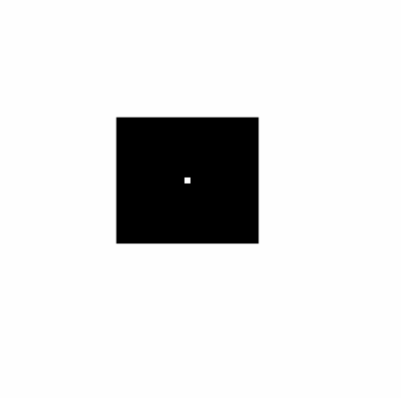


Gráfico 4-8. Cálculo del centro de masa

Gráfico 4-9. Figura irregular

Gráfico 4-10. Cálculo del centro de masa

Gráfico 4-11. Figura irregular

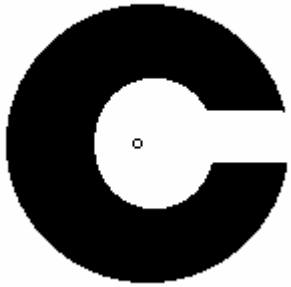


Gráfico 4-12. Cálculo del centro de masa

Gráfico 4-13. Figura irregular

Gráfico 4-14. Cálculo del centro de masa

Se puede observar que en algunas situaciones el centro de masa se ubica fuera de la figura (Gráfico 4-11y Gráfico 4-12)

También es importante destacar que la ubicación del centro de masa es independiente a la orientación del objeto (Gráfico 4-13 y Gráfico 4-14).

Luego de observar los ejemplos podemos concluir que el centro de masa es una buena medida de las coordenadas ocupadas por un objeto en la escena. Una propiedad muy importante es que su cálculo es independiente de la orientación que tenga el objeto. El centro de masa es relativo al objeto, por lo cual siempre está ubicado en la misma posición.

4.4 Armado de las trayectorias

4.4.1 Introducción

Una vez encontrados los objetos relevantes de la escena, y obtenidas las coordenadas que representan sus posiciones en cada instante del recorrido, debe resolverse el problema de seguimiento y estimación de las trayectorias realizadas por dichos objetos dentro de la escena.

Debemos diferenciar claramente los dos problemas mencionados

- Seguimiento de las trayectorias de los objetos
- Predicción de trayectorias futuras de los objetos

Por seguimiento de trayectorias hacemos referencia a la identificación del recorrido realizado por cada objeto en la secuencia de imágenes analizadas.

La predicción se realizará una vez que se han armado las trayectorias de los objetos de la escena. Se utiliza la información del recorrido previo de un objeto para predecir sus futuros movimientos.

4.4.2 Seguimiento de trayectorias

El seguimiento se realiza a partir de la secuencia de imágenes (frames) que representa una escena con objetos en movimiento en un período de tiempo. Consiste en identificar la trayectoria realizada por cada uno de los objetos.

Surge de inmediato el primer problema que se debe resolver, que es el de la identificación de las entidades en movimiento de la escena. Un objeto encontrado en un frame debe ser reconocido como el mismo objeto en los frames subsiguientes. Se establece entonces una correspondencia entre los objetos encontrados en la secuencia de frames. El proceso de correspondencia consiste en identificar el mismo objeto en uno o más cuadros y determinar los cambios en su ubicación. Realizando la correspondencia de los objetos en la secuencia de frames ordenada temporalmente se determina el movimiento de cada uno de ellos dentro de la escena.

El siguiente gráfico (Gráfico 4-15) ejemplifica el proceso de correspondencia en tres frames consecutivos.

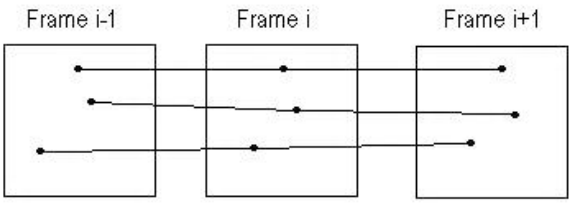


Gráfico 4-15. Correspondencia de objetos en frames consecutivos

Si los objetos a los cuales se les está realizando el seguimiento poseen características que permiten identificarlos y diferenciarlos entre ellos sin confusión, el proceso de correspondencia es inmediato, ya que en cada frame los objetos se reconocen por sus atributos. Entre las características que pueden permitir dicha identificación podemos mencionar, el tamaño, la forma, la intensidad luminosa promedio del objeto, el perímetro, etc. Tales cualidades pueden ser determinadas durante el proceso de segmentación.

Si por el contrario, los objetos no poseen características que permitan distinguirlos entre sí, debe realizarse la correspondencia en función de la posición ocupada por los objetos en la escena. Es decir, dado un conjunto de objetos encontrados en el frame i y otro conjunto encontrado en el frame i+1 se realizará una correspondencia entre los objetos del primer conjunto y los del segundo basándose en sus coordenadas en cada frame y en ciertas características y suposiciones acerca del movimiento de los objetos.

Un enfoque alternativo es una combinación de los dos anteriores, es decir, realizar la correspondencia teniendo en cuenta cualidades de los objetos como así también la ubicación buscando movimientos consistentes.

Se puede observar claramente que la forma de realizar el seguimiento depende de las características de cada problema en particular. Fundamentalmente influye el hecho de que los objetos a seguir posean atributos que permitan distinguirlos unívocamente o por el contrario, que las características de los objetos no sean lo suficientemente distintivas para poder hacerlo.

En este trabajo se analiza la situación en la cual la correspondencia se realiza en función de las posiciones de los objetos, sin tener en cuenta características tales como forma, tamaño u orientación del objeto. Este enfoque es particularmente útil en escenarios donde las entidades con las cuales se trabaja son similares entre sí.

4.4.2.1 Correspondencia

Ya realizado el proceso de reconocimiento de los objetos de un frame, se dispone de una lista de coordenadas que representan las posiciones de los objetos encontrados.

El problema a resolver es el de relacionar un punto pi = (x_i,y_i) de un frame con un punto $p_j = (x_j,y_j)$ del siguiente frame en la secuencia de frames ordenada temporalmente. La disparidad entre los puntos está dada por el vector de desplazamiento entre ambos:

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

Para resolver el problema de correspondencia, es necesario comprender y poder responder las siguientes cuestiones:

- ¿Qué criterio se utiliza para afirmar que dos puntos de dos frames consecutivos se corresponden?
- ¿Cuáles son las características que se comparan en el proceso de correspondencia?
- ¿Existen limitaciones para los vectores de desplazamiento?
- ¿Qué errores pueden ser cometidos en el proceso de correspondencia?

Las siguientes propiedades pueden ayudar a resolver las dos primeras cuestiones planteadas.

- *Similitud*: La similitud es una medida de cuánto dos puntos en dos frames diferentes se asemejan entre sí. Los puntos están representando objetos encontrados en una escena, por lo tanto la similitud va a estar dada por la similitud entre los objetos. La misma se puede determinar a partir de las características encontradas durante el proceso de segmentación. Esta propiedad debe ser tenida en cuenta cuando los objetos poseen características que permiten identificarlos entre sí.
- *Consistencia en el movimiento*: Debido a la inercia, el movimiento de una entidad física no puede cambiar abruptamente. Considerando que la proyección de una trayectoria tridimensional "suave" también lo es en el plano de una imagen de dos dimensiones, se puede afirmar que si las imágenes que se adquieren de la escena se obtienen con una frecuencia tal que entre dos frames consecutivos, el movimiento de los objetos sea leve, entonces para la mayoría de los objetos

físicos las imágenes van a reflejar la suavidad en 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 muestran dos posibles trayectorias. Es claro que la trayectoria ilustrada en (Gráfico 4-17) tiene una mayor consistencia en el movimiento que la trayectoria (Gráfico 4-17). En el primer caso el recorrido es más suave y armonioso, lo que hace suponer que es un recorrido que se asemeja a la realidad. No así el segundo, el cual carece en absoluto de consistencia en el movimiento.

Gráfico 4-17. Trayectoria suave

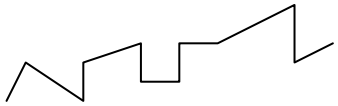


Gráfico 4-17. Trayectoria brusca.

Basándose en la propiedad de consistencia en el movimiento se pueden establecer límites para los vectores de desplazamiento. El límite superior que se establezca, dmax, determinará la distancia máxima que se puede desplazar una entidad de un frame al siguiente. Dicho límite depende fundamentalmente de la frecuencia de muestreo y de la velocidad en el movimiento de los objetos. La elección del límite debe realizarse en forma cuidadosa. Si el límite es muy pequeño, se corre el riesgo de identificar a un mismo objeto como dos objetos distintos si su desplazamiento entre dos frames supera el límite. Por otro lado, si el valor es muy alto, puede corresponderse erróneamente un objeto de un frame con un objeto diferente del siguiente frame, por estar dentro del límite.

4.4.2.2 Posibles errores en el proceso de correspondencia

Básicamente son dos los errores que se pueden cometer durante el proceso de correspondencia:

- Reconocer a dos objetos diferentes como un mismo objeto.
- Reconocer como objetos diferentes al mismo objeto.

Si la correspondencia se realiza basándose en la propiedad de consistencia en el movimiento, el primer error puede ocurrir cuando dos objetos se encuentran muy próximos entre sí. El segundo error puede deberse a la utilización de un límite incorrecto para los vectores de desplazamiento. Si el vector de desplazamiento que representa al movimiento de un objeto en dos frames consecutivos es mayor al límite establecido, el objeto será reconocido como dos objetos diferentes.

Si la correspondencia se realiza basándose en la similitud de ciertas características de los objetos, el primer error puede ocurrir si en la escena hay mas de un objeto con cualidades similares. El segundo error puede ocurrir si las características de un objeto varían sensiblemente de un frame al siguiente. Un ejemplo de esto es un cambio brusco en la iluminación, que provoque que características tales como la luminosidad promedio del objeto, cambien sensiblemente. La mala iluminación puede perturbar también al proceso de segmentación en el momento de determinar los píxeles que corresponden a un objeto, afectando el cálculo de las propiedades de forma, superficie, perímetro, etc.

4.4.2.3 Suposiciones acerca del movimiento de los objetos

Para resolver el problema de seguimiento de las trayectorias realizadas por un conjunto de entidades en movimiento dentro de una escena, es necesario establecer restricciones basadas en la naturaleza de los objetos y sus movimientos. Si la frecuencia de muestreo es relativamente alta en comparación a la velocidad del movimiento de los objetos, y teniendo en cuenta la propiedad de consistencia de movimiento mencionada anteriormente, se deben cumplir las siguientes tres propiedades:

- La ubicación de una entidad en un frame es cercana a la ubicación en el frame siguiente.
- La velocidad de una entidad permanece relativamente sin cambios de un frame al siguiente.
- La dirección en el movimiento de una entidad permanece relativamente sin cambios de un frame al siguiente.

4.4.2.4 Función de desviación de trayectoria

Para formalizar los conceptos presentados hasta el momento, es necesario plantear una función que permita evaluar las propiedades del movimiento de una entidad. La función deberá servir para determinar cuan consistente es el recorrido de la entidad.

La idea es armar una función que reciba una trayectoria y retorne un valor directamente proporcional al grado de desviación del camino. Dicho de otra forma, el valor de retorno será inversamente proporcional a la suavidad de la trayectoria. A dicha función la llamamos *función de desviación.*

Para entender mejor la *función de desviación* de trayectoria consideremos los recorridos de mostrados anteriormente (Gráfico 4-17 y Gráfico 4-17)

El primer recorrido (Gráfico 4-17) representa una curva suave, mostrando una mayor consistencia que el segundo recorrido (Gráfico 4-17). Por lo tanto, la función de desviación

debe retornar un valor menor para el primer recorrido que para el segundo. Esto indica que el segundo recorrido tiene una mayor desviación que el primero.

4.4.2.5 Función de desviación local

Definimos a una trayectoria T, a lo largo de n frames consecutivos, como una sucesión de n puntos.

$T = P_1, P_2,...., P_n$

Para definir la función de desviación de trayectoria, vamos a basarnos en una función que llamaremos *función de desviación local* (fdl), cuyo comportamiento es similar al de la función de desviación de la trayectoria. Al igual que ella, sirve para determinar la desviación de la trayectoria, pero en un punto Pi. Para evaluar la desviación local en el punto P_i se consideran, ademas de P_i , los puntos P_{i-1} y P_{i+1}. Al igual que la función planteada anteriormente, el valor de retorno es directamente proporcional al grado de desviación de la trayectoria en ese punto. Esta función es útil para medir la suavidad de una trayectoria que pasa por esos tres puntos.

A continuación se establecen las pautas que debe cumplir la *función de desviación local* fdl, la cual se basa en los puntos P_{i-1}, P_i, P_{i+1} y determina la desviación del trayecto formado por esos tres puntos.

- Debe estar normalizada al rango [0,1]. Cuanto más cercano a 0 sea el valor, mayor suavidad tendrá el recorrido que pasa por esos puntos. Para un recorrido carente de suavidad en el movimiento, el valor de retorno debe ser cercano a 1.
- Debe tener un valor de respuesta alto a los cambios bruscos de velocidad, así como a los cambios bruscos de dirección.
- Debe responder de igual manera a la misma desviación angular, sin importar la dirección del movimiento. Por ejemplo para los recorridos mostrados a continuación, la función debe retornar el mismo valor.

Gráfico 4-18. Movimientos con igual desviación angular

- Debe responder de igual forma a los cambios debidos a la aceleración.

La fdl debe ser sensible tanto a las variaciones de dirección como a las de velocidad.

Sean P_{i-1}, P_i, P_{i+1} tres puntos correspondientes a las posiciones ocupadas por un objeto en tres frames consecutivos de su trayectoria. Dichos puntos determinan dos vectores de desplazamiento, que representan dos desplazamientos consecutivos del objeto en cuestión.

El módulo de un vector de desplazamiento es una medida de la distancia recorrida, y nos da una aproximación de la velocidad en un punto de la trayectoria. El ángulo del vector es una medida de la dirección en ese punto de la trayectoria.

Por lo tanto, para determinar la desviación de la trayectoria en el punto P_i, pueden compararse los vectores de desplazamiento $P_{i-1}\ P_i$ y $P_i\ P_{i+1}$. Cuanto más similares sean los vectores de desplazamiento, mayor suavidad tendrá el recorrido.

Veamos los conceptos dados con ejemplos.

- La siguiente figura muestra los puntos de un recorrido recto, a velocidad constante. Éste es el caso de un movimiento completamente consistente. Los vectores de desplazamiento en éste caso son idénticos.

- La próxima figura muestra también un recorrido recto, pero la velocidad varía de un frame al siguiente. En este caso el movimiento es mantiene la misma dirección, pero cambia la velocidad. Analizando los vectores de desplazamiento podemos ver que los ángulos de los vectores coinciden, indicando una consistencia en la dirección del movimiento. Sin embargo, los módulos de los ángulos son diferentes, indicando una variación importante en la velocidad de la trayectoria que pasa por esos tres puntos.

- En la próxima figura se muestra un recorrido donde la velocidad del desplazamiento se mantiene constante, pero existe una brusca variación en la dirección del movimiento. Se observa que los módulos de los vectores coinciden, pero no así los ángulos.

- Por último vemos un caso donde no existe consistencia en el recorrido, ni respecto a la dirección del movimiento ni a la velocidad. En este caso, los vectores de desplazamientos son muy diferentes en cuanto a módulo y ángulo.

Es claro que una buena función de desviación local deberá retornar un valor muy próximo a 0 para el primer recorrido (indicando ausencia de desviación local en el recorrido); y un valor cercano a 1 para el recorrido del último caso (indicando un alto grado de desviación en el recorrido).

En los restantes casos deberá retornar valores intermedios, más cercanos a 1 cuanto más desviación se considere que exista en el recorrido.

Estamos entonces en condiciones de armar una *función de desviación local* que tenga en cuenta los aspectos analizados hasta el momento. La función estará compuesta por dos términos, uno de ellos será sensible a las variaciones de dirección, mientras que el otro será sensible a las variaciones de velocidad.

Sean P_{i-1}, P_i, P_{i+1} tres puntos consecutivos de la trayectoria de un objeto. La desviación de la trayectoria que pasa por los tres puntos puede medirse con la siguiente función fdl:

$$fdl (P_{i-1}, P_i, P_{i+1}) = w_1 * variacionDir (P_{i-1}, P_i, P_{i+1}) + w_2 * variacionVel (P_{i-1}, P_i, P_{i+1})$$

Los términos variacionDir y variacionVel son dos funciones normalizadas al rango [0..1] que retornan un valor directamente proporcional a la variación de dirección y de velocidad respectivamente. Las constantes w_1 y w_2 son pesos seleccionados entre 0 y 1, cumpliendo que $w_1 + w_2 = 1$. Modificando los valores de w_1 y w_2, se logra dar mayor importancia a alguna de las variaciones.

4.4.2.6 Función de variación de dirección

En esta sección se arma *función de variación de dirección*, la cual sirve para determinar cuánto cambio existe en la dirección de un recorrido a lo largo de tres frames. La función está normalizada al rango [0..1].

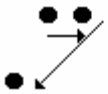
Sean P_{i-1}, P_i, P_{i+1} los tres puntos en cuestión. Llamamos v_1 y v_2 a los dos vectores de desplazamiento que determinan esos tres puntos. Es decir

$v_1 = P_{i-1}\ P_i$

$v_2 = P_i\ P_{i+1}$

La variación de dirección del recorrido está directamente relacionada con la diferencia en el ángulo de los vectores. Dicho de otra forma, la variación de dirección es mayor cuanto mayor sea el ángulo que forman los dos vectores de desplazamiento.

Cuando el ángulo que forman los dos vectores de desplazamiento es 0, no existe ninguna variación en la dirección del movimiento. A medida que el ángulo crece, se tiene una mayor variación de dirección. Esto se ilustra en las siguientes figuras

Recorrido	**Ángulo**	
	0º	La dirección del movimiento no sufre variación.
	45º	Hay un cierto grado de variación de dirección
	90º	Existe un grado alto de variación en cuanto a dirección
	135º	Existe un grado muy alto de variación en cuanto a dirección
	180º	El caso de mayor variación posible en cuanto a dirección.

A continuación se dan dos posibles funciones de variación de dirección. Sea α el ángulo que forman los dos vectores de desplazamiento $\vec{v}_1$ y $\vec{v}_2$ entre sí.

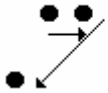
$$a)\ fcionVarDir1 = \frac{1-\cos(\alpha)}{2}$$

La función cos(α) decrece desde 1 hasta -1 en el rango de ángulos que van de 0 a Pi. Cuando el ángulo es 0, la función fcionVarDir1 vale 0, indicando la ausencia de cambios de dirección. A medida que el ángulo que forman los dos vectores de desplazamiento es mayor, la función crece hasta llegar a 1, su máximo valor. Esto ocurre cuando el ángulo que forman los vectores es Pi, el mayor posible.

$$b)\, fcionVarDir2 = \frac{\alpha}{pi}$$

Igual que la función anterior, fcionVarDir2 vale 0 cuando el ángulo es igual a 0 y crece hasta llegar a 1 a medida que el ángulo aumenta.

La siguiente tabla muestra el valor de las dos funciones dadas utilizando los ejemplos planteados anteriormente.

Recorrido	Ángulo (α)	fcionVarDir1	fcionVarDir2
	0º	0	0
	45º	0.14	0.25
	90º	0.5	0.5
	135º	0.85	0.75
	180º	1	1

4.4.2.7 Función de variación de velocidad

El segundo término de la función de desviación local debe ser sensible a las variaciones de velocidad en el trayecto formado por los puntos P_{i-1}, P_i y P_{i+1}. Los valores del término deben estar dentro del rango [0..1]. Un valor 0 indicará la ausencia absoluta de cambios de velocidad, mientras que un valor cercano a 1 indicará un gran cambio en la velocidad del trayecto.

Anteriormente vimos que el módulo del vector que se forma al unir dos puntos consecutivos de la trayectoria es una medida de la velocidad en ese desplazamiento en la trayectoria. El término de variación de velocidad entonces debe tener en cuenta los módulos de los dos vectores de desplazamiento: $v_1 = P_{i-1}\, P_i$ y $v_2 = P_i\, P_{i+1}$.

A continuación se presentan posibles funciones de variación de velocidad. Sean m_1 y m_2 los dos módulos de los vectores de desplazamiento.

$$m_1 = \left| P_{i-1}\ P_i \right|$$
$$m_2 = \left| P_i\ P_{i+1} \right|$$

$$a)\ fcionVarVel1 = 1 - \frac{2.menor(m_1, m_2)}{m_1 + m_2}$$

Cuando los dos módulos son iguales, la fórmula dada vale 0, indicando de esta forma la ausencia absoluta de cambios de velocidad.

A medida que la diferencia entre los módulos crece, el término $\frac{2.menor(m_1, m_2)}{m_1 + m_2}$ toma valores más chicos, con lo cual, la fórmula entera toma valores cercanos a 1.

Veamos el valor que tomará la función en el caso que el módulo del vector más grande es el doble del módulo del vector más chico ($m_1 = 2 * m_2$).

La fórmula en este caso quedará:

$$fcionVarVel1 = 1 - \frac{2.m_2}{2m_2 + m_2} = 1 - \frac{2}{3} = \frac{1}{3}$$

Esta fórmula tiene una propiedad poco deseable que es que no tiene en cuenta las magnitudes de m_1 y m_2.

En el ejemplo planteado, el valor del término es 1/3 si se cumple que el módulo de uno de los vectores de desplazamiento es el doble que el otro.

Si por ejemplo, los vectores de desplazamiento tienen módulos m_1=2 y m_2=4, la función de desviación de velocidad da el mismo resultado que si los módulos valen m_1=10 y m_2=20, o m_1=30 y m_2=60. En cualquiera de los casos el valor de la función es 1/3.

Recordemos que durante el procesamiento de un frame, tanto en su captura como en la segmentación y el cálculo de las coordenadas de los objetos existen factores que introducen ruido que afectan a los resultados obtenidos.

Supongamos que los módulos de los vectores de desplazamiento del movimiento observado de un objeto son m_1=2 y m_2=4. Es probable que debido al ruido existente en alguno de los pasos del proceso que permite llegar a la determinación de las coordenadas del objeto dentro de la escena esos valores no representen en forma precisa los desplazamientos reales del objeto. Si los valores reales de los módulos son m_1=3 y m_2=3, en el movimiento real del objeto no existe desviación de velocidad.

Supongamos ahora que los módulos de los vectores de desplazamiento son m_1=10 y m_2=20. A pesar de que también puede existir ruido, es poco probable que los valores reales de

m_1 y m_2 tengan valores próximos. Eso sólo sucedería si el ruido fuera muy grande. Por lo tanto, en este caso existe un cierto grado de variación de velocidad en la trayectoria.

La FcionDesvVelocidad1 devuelve el mismo resultado en ambos casos, a pesar que como vimos, es altamente probable que exista una desviación mayor en el segundo caso. Esto se debe a que el ruido afecta a la fórmula mucho más cuando los vectores de desplazamiento tienen valores chicos.

Es deseable entonces que para valores pequeños de m_1 y m_2, la fórmula no castigue las diferencias entre los módulos de la misma manera que para valores grandes, a pesar de que la proporción entre m_1 y m_2 sea la misma.

La siguiente función tiene en cuenta la magnitud de la diferencia de los módulos. Recordemos que d_{max} es el máximo desplazamiento tolerado en el movimiento de un objeto en dos frames consecutivos, lo que es equivalente al valor máximo que puede tener el módulo de un vector de desplazamiento.

$$b) FcionDesvVelocidad2 = \frac{|m_1 - m_2|}{d_{max}}$$

Debido a que $m_1 < d_{max}$ y $m_2 < d_{max}$ se cumple que $|m_1 - m_2| < d_{max\ 1}$. Por lo tanto la fórmula da valores entre 0 y 1. Cuando m1=m2 la fórmula vale 0, acercándose a 1 a medida que la diferencia entre los módulos se incrementa.

4.4.2.8 Desviación de la trayectoria

Una vez planteadas las funciones de variación de dirección y de velocidad, retomamos la fórmula original de la función de desviación local fdl, la cual es una medida de la desviación de la trayectoria que pasa por los puntos P_{i-1}, P_i, $P_{i+1.}$

fdl (P_{i-1}, P_i, P_{i+1}) = w_1 . variacionDir + w_2 . variacionVel

Sean

$v_1 = P_{i-1}\ P_i$

$v_2 = P_i\ P_{i+1}$

α_1 el ángulo que forman v_1 y v_2

$m_1 = |m_1|$

$m_2 = |m_2|$

$$\text{fdl}\ (P_{i-1}, P_i, P_{i+1}) = w_1 \frac{1 - \cos(\alpha)}{2} + w_2 \frac{|m_1 - m_2|}{d_{max}}$$

Esta función devuelve valores en el rango [0..1] dando una medida de la desviación del trayecto que pasa por los puntos P_{i-1}, P_i, P_{i+1}. Dicho de otra manera, la función da una medida de la suavidad de una trayectoria que pasa por esos tres puntos.

Sea $T = P_1, P_2,...., P_n$ una secuencia de puntos que representan una trayectoria a lo largo de n frames. La desviación de la trayectoria completa se calcula como la suma de las desviaciones locales en cada punto a lo largo de toda la trayectoria.

$$desv(T) = \sum_{i=2}^{n-1} \text{fdl}\,(P_{i-1}, P_i, P_{i+1})$$

Si en la escena existen *m* objetos en movimiento, a lo largo de los n frames se tienen *m* trayectorias, $T_1, T_2,...., T_m$.

La desviación global de toda la escena a lo largo de los *n* frames se calcula con la siguiente fórmula:

$$desvTotal = \sum_{i=1}^{m} \text{desv}(T_i)$$

4.4.2.9 Solución óptima al problema de correspondencia

Analizando los conceptos planteados previamente, se concluye que la solución óptima al problema de la correspondencia de puntos consiste en encontrar el conjunto de trayectorias que minimice la desviación total. De esta forma se obtienen trayectorias con la mayor suavidad en el movimiento posible.

Para resolver el problema de correspondencia minimizando la desviación total, es necesario analizar todas las combinaciones posibles de correspondencia de puntos a lo largo de los *n* frames, seleccionando aquella combinación que produzca la menor desviación global. Para ello se necesita conocer de antemano todos los puntos en los *n* frames, lo cual no es posible en el problema que se plantea en este trabajo, ya que los frames se procesan en tiempo real y las trayectorias deben armarse a medida que se va incorporando la información de los objetos detectados en cada uno.

4.4.2.10 Correspondencia a medida que se procesan los frames

Es necesario resolver el problema de armado de trayectorias a medida que se van incorporando nuevos frames. A partir de cada nueva imagen que se procesa en tiempo real, se deben ampliar las trayectorias existentes realizando una correspondencia entre sus últimos

puntos y los puntos que determinan las coordenadas de los objetos encontrados en el nuevo frame. A continuación se presenta un posible algoritmo que resuelve el problema de la correspondencia a medida que se procesan los frames.

4.4.2.11 Algoritmo de correspondencia

Inicialmente se crea una trayectoria por cada objeto encontrado en el primer frame. Luego, a medida que se van procesando los nuevos frames se enlazan los objetos encontrados en cada uno a las trayectorias existentes. A continuación se muestra el esqueleto del algoritmo.

```
Armar una trayectoria por cada objeto encontrado en el frame 1

Loop
      Adquirir nuevo frame
      Encontrar los objetos
      Actualizar las trayectorias con los nuevos objetos (*)
```

En la última línea del loop del algoritmo planteado (*) es donde se realiza el proceso de correspondencia. Considerando la función de desviación local, la solución al problema consiste en corresponder el último punto de cada trayectoria T_k, con un punto del nuevo conjunto, de tal forma que globalmente se minimice dicha función.

A continuación se explica una forma de resolver el problema de actualización de las trayectorias con los objetos hallados en un nuevo frame. En un principio se supone que el mismo conjunto de objetos es encontrado a lo largo de la secuencia completa de frames. Luego le haremos las modificaciones necesarias para que el algoritmo se adapte a cualquier circunstancia.

Sean $T_1,...,T_m$ las trayectorias armadas hasta el momento de procesar el frame i. Llamemos C_{i+1} al conjunto de puntos que determinan las coordenadas de los objetos hallados en el frame i+1.

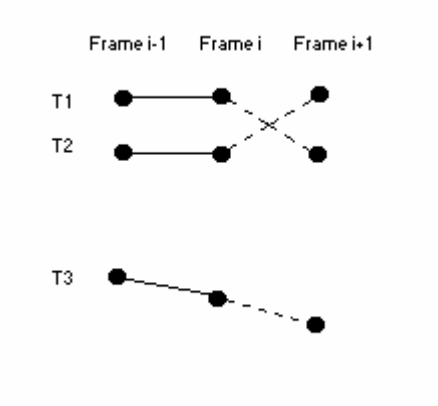
Sea d_{max} el límite superior que se establece para los vectores de desplazamiento. Es decir, la máxima distancia tolerada que un objeto puede avanzar entre dos frames consecutivos. Este valor está estrechamente relacionado con la frecuencia de muestreo y la velocidad del movimiento de los objetos de la escena.

El objetivo es extender las trayectorias $T_1,...,T_m$ asociándole a cada una un punto del nuevo conjunto C_{i+1}, de tal manera que se minimice la desviación global de las trayectorias.

Inicialmente, a cada trayectoria T_k se le asocia un punto del conjunto C_{i+1} que esté a distancia menor a d_{max}. En caso de múltiples alternativas se resuelve en forma arbitraria.

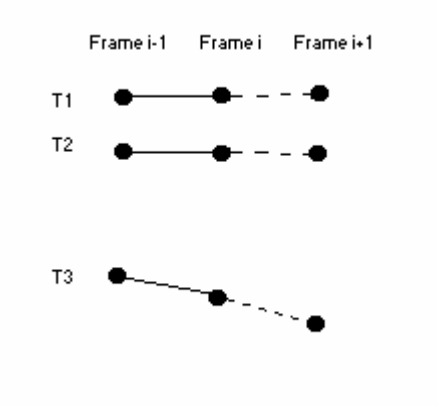
Luego que cada trayectoria tiene enlazado un punto del nuevo conjunto, se realiza lo que llamamos ciclo de intercambio. Dicho ciclo busca intercambiar entre las trayectorias los últimos puntos recién asociados de tal forma que se minimice la desviación de las trayectorias.

Para entender mejor la utilidad del ciclo de intercambio, se plantea a continuación un ejemplo.



El gráfico muestra una escena con tres objetos a lo largo de tres frames. Las líneas sólidas muestran las trayectorias ya armadas hasta el momento, las cuales llegan hasta el frame i. Las líneas punteadas muestran la correspondencia realizada con los nuevos puntos del frame i+1. Vamos a suponer que cualquiera de los dos primeros puntos del nuevo frame puede asociarse a las dos primeras trayectorias, T_1 y T_2, debido a que se encuentran a una distancia menor a d_{max}.

A simple vista se observa que la correspondencia realizada no es la óptima, ya que provoca una mayor desviación en los trayectos que la correspondencia mostrada a continuación.



El ciclo de intercambio resuelve este tipo de situaciones intercambiando los puntos entre las trayectorias involucradas buscando minimizar la función de desviación.

4.4.2.12 Función de ganancia de intercambio de puntos entre trayectorias

Para poder resolver el ciclo de intercambio, es necesario, dadas dos trayectorias T_k y T_h, establecer algún criterio para determinar si es conveniente intercambiar sus últimos puntos. En ésta sección se plantea una forma de evaluar los beneficios de un intercambio de los últimos puntos de dos trayectorias.

Sean P^k_{i-1} , P^k_i , P^k_{i+1} los puntos de la trayectoria T_k correspondientes a los frames i-1, i e i+1 respectivamente; y P^h_{i-1} , P^h_i , P^h_{i+1} los puntos de los mismos frames correspondientes a la trayectoria T_h. Se necesita una función que determine si es conveniente intercambiar los últimos puntos de tal manera que la trayectoria T_k quede formada por los puntos P^k_{i-1} , P^k_i , P^h_{i+1} y la trayectoria T_h por los puntos P^h_{i-1} , P^h_i , P^k_{i+1}.

Utilizando la función de desviación local planteamos ahora la función de *ganancia de intercambio* de puntos entre las trayectorias. Esta función toma como argumentos dos trayectorias y devuelve un valor que determina cual es la ganancia de intercambiar los últimos puntos entre las trayectorias.

Definimos:

$$G_{kh}{}^{i+1} = dsi_{kh}{}^{i+1} - dci_{kh}{}^{i+1}$$

Donde $dsi_{kh}{}^{i+1}$ (desviación sin intercambios) es una medida de la desviación de ambas trayectorias a lo largo de los frames i-1, i e i+1 y $dci_{kh}{}^{i+1}$ (desviación con intercambios) es una medida de la desviación de ambas trayectorias con los puntos correspondientes al frame i+1 intercambiados.

$$dsi_{kh}{}^{i+1} = fdl(P^h{}_{i-1}, P^h{}_i, P^h{}_{i+1}) + fdl(P^k{}_{i-1}, P^k{}_i, P^k{}_{i+1})$$

$$dci_{kh}{}^{i+1} = fdl(P^h{}_{i-1}, P^h{}_i, P^k{}_{i+1}) + fdl(P^k{}_{i-1}, P^k{}_i, P^h{}_{i+1})$$

La función de ganancia $G_{kh}{}^{i+1}$ considera la diferencia entre las dos desviaciones. Si dicha diferencia es positiva existe una mayor desviación si no se intercambian los puntos. Si la diferencia es negativa, la desviación es mayor si los últimos puntos de las trayectorias se intercambian. Si la función es positiva, cuanto más grande sea su valor, mejor será realizar el intercambio de puntos. Por lo tanto, la función es una medida de cuan conveniente es realizar el intercambio de puntos entre las trayectorias.

4.4.2.13 Ciclo de intercambio

Planteada la función de ganancia en el intercambio de puntos entre trayectorias, volvemos al algoritmo de correspondencia, resolviendo el ciclo de intercambio de puntos.

Sea m el número de trayectorias armadas a lo largo de los primeros i frames, y C_{i+1} el conjunto de puntos encontrados en el frame i+1.

El primer paso consiste en asociar a cada trayectoria T_k, un punto del conjunto C_{i+1} que esté a distancia menor a d_{max}, resolviendo arbitrariamente en caso de múltiples alternativas. Hecho esto, tenemos que cada trayectoria se extiende hasta el frame i+1. A continuación se presenta el algoritmo de intercambio. Dado que la distancia recorrida por un objeto en dos frames consecutivos no debe exceder el valor de d_{max} se evalúa el intercambio entre aquellos pares de trayectorias que cumplen la condición que al cambiar entre ellas sus últimos puntos, la distancia máxima entre dos puntos consecutivos no supere dicho valor.

El objetivo del ciclo de intercambio es minimizar la desviación de las trayectorias.

```
FOR t1 = 1 hasta m-1   // iteración sobre las trayectorias
      FOR t2 = t1 + 1 hasta m
            Si se cumple la restricción de dmax (*)
                  calcular G_{t1,t2}^{i+1}

Tomar Gmax = el par G_{t1,t2}^{i+1} de mayor ganancia
Si Gmax > 0
      Intercambiar los puntos entre las trayectorias que determinan
      la ganancia máxima

Repetir el ciclo hasta que no se produzcan mas intercambios
```

Este ciclo itera a través de todas las trayectorias, buscando si existe alguna ganancia al intercambiar puntos entre trayectorias. De ser así, intercambia los últimos puntos entre las trayectorias que determinen la mayor ganancia.

Si hubo algún intercambio el ciclo debe repetirse hasta que no haya más intercambios, lo cual sucede cuando la correspondencia realizada es la óptima en cuanto a la función de ganancia utilizada.

La línea marcada con (*) verifica antes de intentar el intercambio, que sea posible intercambiar los puntos entre el par de trayectorias que se está considerando. Lo que se controla es que en caso de intercambiarse los puntos, se cumpla la restricción de que la distancia entre dos puntos consecutivos de una trayectoria no exceda el valor de dmax.

Es importante mencionar que el intercambio de puntos tiene sentido cuando las trayectorias se extienden a lo largo de tres frames por lo menos, ya que la desviación se calcula en función de tres puntos consecutivos de la trayectoria. En caso de tener trayectorias cuya longitud sea menor a tres no se efectúa el ciclo de intercambio.

4.4.2.14 Problemas en el algoritmo de correspondencia

El algoritmo planteado considera un escenario donde los mismos objetos son encontrados a lo largo de toda la secuencia de frames. En la práctica, esta situación no es la real. A lo largo de una secuencia de imágenes, es posible que algunos objetos desaparezcan en forma parcial o total. Puede ocurrir algunas de las siguientes situaciones:

- Un objeto puede no ser encontrado en un frame determinado. Esto puede deberse a algún error en la etapa de búsqueda de los objetos, causado, por ejemplo, por ruido o malas condiciones de iluminación. Sin embargo, el objeto puede aparecer nuevamente en los frames siguientes.
- En un frame determinado, en la salida del proceso de detección puede haber un punto que en realidad no corresponde a ninguno de los objetos de la escena. Esta introducción de falsa información puede ser originada por los mismos problemas mencionados en el punto anterior.
- Un objeto puede dejar de ser capturado a partir de un frame, y de allí en adelante, debido a que se excedió de los límites de la región analizada.
- A partir de un determinado frame puede aparecer un nuevo objeto en la escena.

La cantidad de objetos encontrados en cada frame a lo largo de toda la secuencia puede no ser la misma. Aún más, en caso de encontrarse la misma cantidad de objetos en cada frame, es probable que los objetos no sean los mismos.

El algoritmo de correspondencia debe modificarse para que considere los casos planteados anteriormente. Debe permitirse el armado trayectorias incompletas, donde falten puntos intermedios por no haber sido detectados en uno o más frames y debe considerarse la situación de tener que crear nuevas trayectorias en frames intermedios de la secuencia.

4.4.2.15 Puntos ausentes

Para solucionar el problema de las trayectorias incompletas, introducimos el concepto de *puntos ausentes*. Un punto ausente es usado para completar la trayectoria de un objeto en un frame donde el objeto no haya sido detectado.

Entonces, una trayectoria T, a lo largo de n frames, es una secuencia de puntos de la forma

$$T = P_1, P_2, ..., P_n$$

donde algunos de los puntos P_i, con i=1..n, pueden ser ausentes.

4.4.2.16 Correspondencia con puntos ausentes y nuevas trayectorias

Es necesario modificar el algoritmo de correspondencia planteado anteriormente, para que permita realizar las siguientes acciones:

- Completar una trayectoria con *puntos ausentes*, en el caso que en un determinado frame la trayectoria no sea asociada a ninguno de los objetos hallados.
- Crear nuevas trayectorias que comiencen en frames intermedios de la secuencia, para el caso que en un frame determinado aparezca un nuevo objeto en la escena.

En el caso que exista falsa información, es decir, que en un determinado frame la salida del proceso de identificación de objetos contenga un punto que no corresponda a ninguno de los objetos de la escena real, el algoritmo formará una nueva trayectoria con dicho punto y la completará con puntos ausentes. Tales trayectorias pueden ser descartadas posteriormente, dejando aquellas trayectorias en donde la mayoría de sus puntos son reales.

Recordemos el esqueleto del algoritmo de seguimiento y armado de trayectorias.

```
Armar una trayectoria por cada objeto encontrado en el frame 1

Loop
      Adquirir nuevo frame
      Encontrar los objetos
      Actualizar las trayectorias con los nuevos objetos (*)
```

Se debe modificar entonces el paso (*), el cual consiste en la actualización de las trayectorias con los objetos encontrados en el nuevo frame.

Sea $T_1,...,T_m$ el conjunto de trayectorias armadas hasta el momento; C_{i+1} el conjunto de puntos que determinan las coordenadas de los objetos hallados en el nuevo frame; y d_{max} el límite superior que se establece para los vectores de desplazamiento. Los pasos a realizar son:

1. A cada trayectoria T_k se le asocia un punto del conjunto C_{i+1} que esté a distancia menor a d_{max}. En caso de múltiples alternativas se resuelve en forma arbitraria (igual que en el algorimo original)
2. A cada trayectoria T_k que no fue asociada a ninguno de los nuevos puntos, asociarle un *punto ausente*.
3. Por cada punto del conjunto C_{i+1} que no haya sido asociado a ninguna trayectoria, crear una nueva trayectoria que comience en ese punto.

4. Realizar el ciclo de intercambio de puntos entre las trayectorias, considerando que ahora pueden existir *puntos ausentes* y trayectorias que comienzan en frames intermedios de la secuencia. El ciclo debe repetirse hasta que no se produzcan más intercambios.
5. Determinar las coordenadas de los nuevos puntos ausentes que existan en las trayectorias.

A continuación se explican en un ejemplo los pasos del algoritmo planteado. Supongamos que se está procesando el frame i+1. Hasta el frame i se tienen armadas 3 trayectorias, T1, T2 y T3. En el frame i+1 se encontraron tres puntos, P1, P2 y P3 que representan tres objetos hallados (Gráfico 4-19).

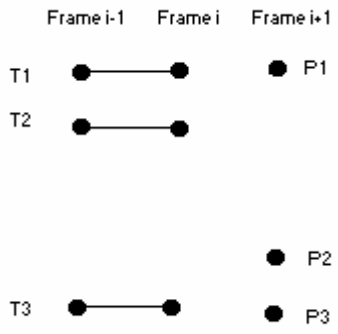


Gráfico 4-19

El paso 1 del algoritmo de correspondencia consiste en asociarle a cada trayectoria, uno de los nuevos puntos que se encuentre a distancia menor a d_{max}.

Supongamos que a las trayectorias T_1 y T_2 sólo puede asociarse el punto P_1 por encontrarse a distancia menor a d_{max}; y a la trayectoria T_3 puede asociarse tanto P_2 como P_3.

La próxima figura muestra las correspondencias realizadas en el paso 1 (Gráfico 4-20).

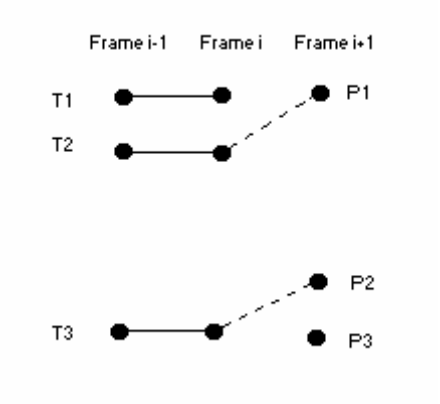


Gráfico 4-20

Observar que la trayectoria T_1 no fue asociada a ninguno de los nuevos puntos, ya que el único punto que se encuentra a distancia menor a d_{max} es P_1, pero este punto fue correspondido a la trayectoria T_2.

Por otro lado, la trayectoria T_3 fue correspondida con el punto P_2, por lo tanto quedó el punto P_3 sin ser correspondido.

El siguiente paso (2) consiste en asociarle un *punto ausente* a cada trayectoria que haya quedado sin ser correspondida. En nuestro ejemplo, la única trayectoria que quedó sin ser correspondida es la T_1, es a ella entonces a la que se le asocia un punto ausente (Gráfico 4-21).

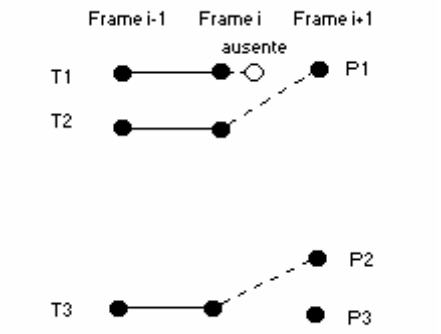


Gráfico 4-21

Una trayectoria con puntos ausentes será tenida en cuenta en el paso 4, que consiste en el intercambio de puntos entre las trayectorias. Por lo tanto, es posible que un punto ausente asociado a una trayectoria, luego del proceso de intercambio pase a formar parte de otra trayectoria. Por lo tanto, en el paso 2, a los puntos ausentes creados no se les establece aún coordenadas (en el gráfico se ubica en tal posición sólo para clarificar). Recién luego del proceso de intercambio, a cada punto ausente se le establecerá sus coordenadas en función de las coordenadas de los restantes puntos de las trayectorias a las cuales fueron asociados.

Luego del paso 3 del algoritmo, se tienen 4 trayectorias, una más que las que existían hasta el momento. La trayectoria número 4 empieza en el frame i+1, y antes de comenzar el proceso de intercambio de puntos entre las trayectorias está formada sólo por el punto P_3.

La siguiente imagen (Gráfico 4-22) muestra el estado de las trayectorias antes de comenzar el paso de intercambio del algoritmo, el cual busca minimizar la desviación local, y es explicado en la próxima sección.

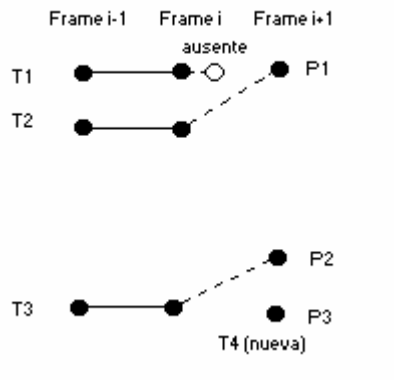


Gráfico 4-22

4.4.2.17 Ciclo de intercambio con puntos ausentes y nuevas trayectorias

Una vez realizados los tres primeros pasos del algoritmo de correspondencia, llega el momento de efectuar el ciclo de intercambio de puntos entre las trayectorias buscando minimizar la desviación local.

A continuación se muestra el pseudocódigo que resuelve el ciclo de intercambio, que fue planteado con anterioridad.

```
FOR t1 = 1 hasta m-1   // iteración sobre las trayectorias
      FOR t2 = t1 + 1 hasta m
            Si se cumple la restricción de dmax (*)
                  calcular G_{t1,t2}^{i+1}

Tomar Gmax = el par G_{t1,t2}^{i+1} de mayor ganancia
Si Gmax > 0
      Intercambiar los puntos entre las trayectorias que determinan
      la ganancia máxima

Repetir el ciclo hasta que no se produzcan mas intercambios
```

Debido a la presencia de puntos ausentes formando parte de las trayectorias y a la posibilidad de crear trayectorias en el medio de la secuencia de frames por aparición de un nuevo objeto, es necesario resolver las siguientes cuestiones:

- Dado que a los *puntos ausentes* no se les establece inicialmente coordenadas, en la línea marcada con (*) ¿cómo se determina si se cumple la restricción de d_{max}, si uno de los puntos considerados es un *punto ausente*?
- ¿Cómo se calcula la función de ganancia en el caso de estar evaluando una trayectoria armada recientemente que no se extiende a lo largo de tres frames?
- ¿Cómo se calcula la función de ganancia de intercambio en el caso de estar considerando puntos ausentes?

Sea T una trayectoria y P un punto del nuevo frame. Para determinar si P se puede corresponder con la trayectoria T, es necesario verificar que la distancia entre P y el último punto de T, llamémoslo U, no exceda el valor de d_{max}. Definimos la función de distancia entonces de la siguiente manera:

$$\text{Distancia}(U,P) = \begin{cases} d_{max} & \text{si P es un punto ausente} \\ \text{distancia eucledian}(U,P) & \text{si P no es un punto ausente} \end{cases}$$

Notar que U puede ser un punto ausente, pero al ya estar asociado a una trayectoria, tiene determinadas sus coordenadas

Al definir la función de distancia de esa forma, cualquier *punto ausente* puede ser asociado a una trayectoria ya que siempre cumple la restricción d_{max}. Esto lleva a que en el algoritmo, una trayectoria que haya sido asociada a un *punto ausente*, será considerada en el ciclo de intercambio, permitiendo que el punto ausente pase a formar parte de otra trayectoria si es conveniente.

La función de ganancia de intercambio de puntos entre trayectorias solo debe evaluarse si al menos una de las trayectorias se extiende a lo largo de los últimos tres frames. No tiene sentido intercambiar puntos entre dos trayectorias que tienen longitud menor que tres, ya que la función de desviación de una trayectoria se calcula considerando sus últimos tres puntos.

Queda entonces definir la función de ganancia de intercambio de puntos entre dos trayectorias k y h en el instante de tiempo i+1, de tal manera que permita la presencia de puntos ausentes y trayectorias nuevas.

Recordamos la definición de las funciones presentadas anteriormente.

Sean

$v_1 = P_{i-1}\ P_i$
$v_2 = P_i\ P_{i+1}$

α_1 el ángulo que forman v_1 y v_2
$m_1 = |m_1|$
$m_2 = |m_2|$

$$fdl(P_{i-1}, P_i, P_{i+1}) = w_1 \frac{1-\cos(\alpha)}{2} + w_2 \frac{|m_1 - m_2|}{d_{max}}$$

$$dsi_{kh}^{i+1} = fdl(P^h_{i-1}, P^h_i, P^h_{i+1}) + fdl(P^k_{i-1}, P^k_i, P^k_{i+1}).$$

$$dci_{kh}^{i+1} = fdl(P^h_{i-1}, P^h_i, P^k_{i+1}) + fdl(P^k_{i-1}, P^k_i, P^h_{i+1}).$$

$$G_{kh}^{i+1} = dsi_{kh}^{i+1} - dci_{kh}^{i+1}$$

La única función que es necesario redefinir para que el algoritmo del ciclo de intercambio de puntos se adapte a la presencia de puntos ausentes y nuevas trayectorias es la fdl, ya que la ganancia está definida en función de ésta.

Definimos entonces la fdl', de la siguiente manera:

$$fdl'(P_{i-1}, P_i, P_{i+1}) = \begin{cases} fdl(P_{i-1}, P_i, P_{i+1}) & \text{si los tres puntos tienen} \\ & \text{determinadas sus coordenadas} \\ 0 & \text{en caso contrario} \end{cases}$$

La nueva función devuelve el mismo valor que la original si los tres puntos son reales o si alguno de ellos es ausente pero tiene determinada las coordenadas; y devuelve 0 en caso que P_{i+1} sea ausente o la trayectoria sea una recién creada (si una trayectoria fue creada en el frame i o i+1 el punto P_{i-1} de la trayectoria no existe).

Recordar que un punto ausente representa la posición de un objeto en un frame que no fue detectado. Veremos más adelante que sus coordenadas se calculan en función de los otros puntos de la trayectoria.

La función de ganancia queda definida entonces en base a la nueva función de desviación local (fdl').

El algoritmo de intercambio de puntos finalmente queda como se muestra a continuación:

```
FOR t1 = 1 hasta m-1   // iteración sobre las trayectorias
```

```
      FOR t2 = t1 + 1 hasta m
            Si se cumple la restricción de dmax y al menos una de las
            trayectorias t1 y t2 tiene longitud mayor a 2
                  calcular G_{t1,t2}^{i+1}

Tomar Gmax = el par G_{t1,t2}^{i+1} de mayor ganancia
Si Gmax > 0
      Intercambiar los puntos entre las trayectorias que determinan
      la ganancia máxima

Repetir el ciclo hasta que no se produzcan mas intercambios
```

4.4.2.18 Resultado del ciclo de intercambio en el ejemplo planteado

Planteado el ciclo de intercambio de puntos con la nueva función de ganancia, retomamos el ejemplo dado.

Hasta antes de comenzar el ciclo de intercambio se tenían 4 trayectorias, armadas de la forma que se muestra en la próxima figura (Gráfico 4-23).

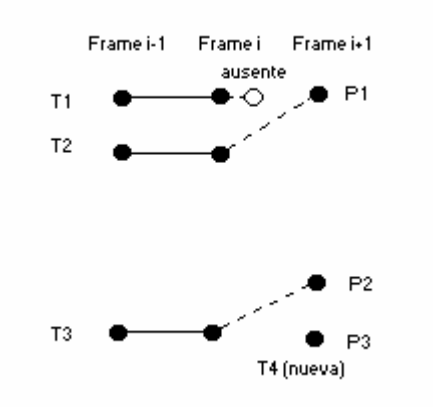


Gráfico 4-23

Luego del ciclo de intercambio las trayectorias quedan como se muestra a continuación (Gráfico 4-24). Se puede observar que se intercambiaron los últimos puntos entre las trayectorias T_1 y T_2 y se realizó un intercambio entre los últimos puntos de las trayectorias T_3 y T_4. Las trayectorias armadas de esta forma, provocan una menor desviación que la que existía antes del ciclo de intercambio.

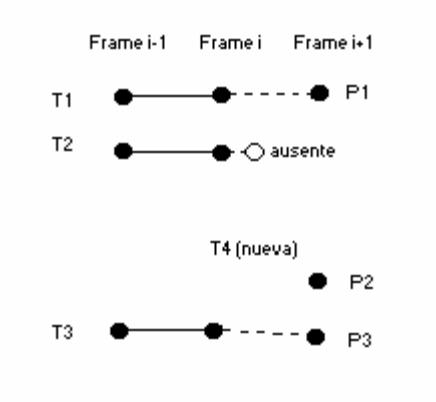


Gráfico 4-24

El punto ausente de la trayectoria T_2 todavía no tiene determinadas sus coordenadas. En la siguiente sección se explica como se determinan las coordenadas de los puntos ausentes.

4.4.2.19 Actualización de las coordenadas de los puntos ausentes

Luego de realizar el ciclo de intercambio, lo único que falta para completar el algoritmo de correspondencia de puntos es darle coordenadas a los nuevos puntos ausentes introducidos en los pasos previos.

Supongamos que estamos procesando el frame k, y sea $T= P_j, P_{j+1} .... P_k$ una trayectoria que comienza en el frame j, con $j \geq 1$ y $j < k-1$, cuyo último punto P_k es un punto ausente. Sean $P_{k-2} = (X_{k-2} , Y_{k-2})$ y $P_{k-1} = (X_{k-1} , Y_{k-1})$ las coordenadas de puntos de la trayectoria correspondientes a los frames k-1 y k-2. Las coordenadas de P_k las establecemos de la siguiente forma:

$$P_k = (X_k , Y_k)$$

donde $X_k = X_{k-1} + (X_{k-1} - X_{k-2})$

$$Y_k = Y_{k-1} + (Y_{k-1} - Y_{k-2})$$

Vemos que las coordenadas del punto ausente se establecen en función de los dos puntos anteriores de la trayectoria. Lo que hacemos es sumarle a las coordenadas del punto anterior P_{k-1} el desplazamiento entre los frames k-2 y k-1. Este cálculo se basa en la propiedad de consistencia en el movimiento, la cual supone que el nuevo desplazamiento debe ser similar al anterior.

En el caso de tener una trayectoria que fue creada en el frame k-1 y en el frame k tiene un punto ausente, no es posible darle coordenadas al punto ausente debido a que la trayectoria tiene un solo punto real. Lo más probable es que ésta trayectoria haya sido creada en el frame k-1 debido a la presencia de un *punto falso*, es decir, un punto que fue detectado en la etapa de

reconocimiento de objetos, que en realidad no se corresponde a ninguno de los objetos reales de la escena. Este tipo de errores se producen debido al ruido. Entonces, aquellas trayectorias creadas en un frame determinado a las cuales se les asocia un punto ausente en el siguiente frame, son eliminadas del conjunto de trayectorias, bajo la suposición de que son *falsas trayectorias*. En el caso que una trayectoria eliminada de esta forma corresponda a un objeto real, en el siguiente frame que se detecte el objeto se comenzará a armar nuevamente la trayectoria. Por lo tanto, se puede eliminar una trayectoria cuyo segundo punto es ausente, ya que en caso de ser una trayectoria real, se comenzará a armar en el siguiente frame.

4.4.2.20 Consideraciones realizadas en el seguimiento de trayectorias

Se analizó el caso genérico donde se tienen varios objetos en la escena. La cantidad de objetos en la secuencia de frames puede variar ya que un objeto puede aparecer o desaparecer de la escena en cualquier momento.

En el caso de una escena con un solo objeto, el seguimiento de trayectorias es trivial, ya que no se necesitaría realizar ningún proceso de correspondencia. Sólo bastaría con encontrar el objeto en cada frame.

Se consideró también que los objetos no poseen características distintivas que permitan identificarlos. En caso contrario, el proceso de correspondencia puede realizarse en función de ciertas características calculadas en el proceso de segmentación, en lugar hacerlo en función de las posiciones ocupadas por el objeto en cada frame. Características que según el caso pueden servir para identificar un objeto unívocamente son, entre otras, el perímetro del objeto, el área, la forma, la intensidad promedio, etc.

4.4.3 Predicción de trayectorias

La salida del proceso de seguimiento de trayectorias es el conjunto de trayectorias realizadas por los objetos encontrados en la escena, a lo largo de toda la secuencia de frames.

Una vez que se han armado las trayectorias de los objetos de la escena, analizaremos el problema de usar dicha información, para poder predecir los futuros movimientos de cada objeto.

El problema de predicción de la trayectoria de un móvil consiste en analizar el recorrido realizado por éste y basarse en él para estimar el recorrido posterior. Se busca estimar de la mejor manera posible la futura trayectoria del móvil.

El error cometido en la estimación de la trayectoria de un objeto depende de diversos factores. En primer lugar, estará influenciado por la calidad de la información acerca de la trayectoria previa del objeto que se usa como base para la estimación. Es claro que cuanto más exacta sea la información del recorrido previo se podrán obtener resultados más precisos. Por otro lado, cuanto mayor conocimiento se disponga acerca de las características del movimiento de un objeto, mejor será la estimación. Si un objeto se mueve realizando una trayectoria

circular y se estima el recorrido futuro suponiendo que el objeto realiza una trayectoria lineal, la estimación resultará en un fracaso. Por lo tanto es importante el conocimiento que se disponga acerca de las características del movimiento para poder modelarlo con la mayor exactitud posible.

4.4.3.1 Problema a resolver

Dada una secuencia de puntos que representa las coordenadas observadas en períodos regulares de tiempo del recorrido realizado por un objeto en movimiento, se pretende estimar el futuro recorrido del objeto.

Sea P_1, P_2,....P_n la serie de puntos que representan las posiciones de un objeto en los instantes de tiempo t_1, t_2...t_n. Interesa estimar m posiciones futuras P_{n+1} , P_{n+2},....P_{n+m} correspondiente a los m instantes de tiempo futuro t_{n+1}, t_{n+2}...t_{n+m} .

Debe tenerse en cuenta que las coordenadas que se disponen del recorrido del objeto fueron calculadas durante las primeras etapas del sistema, a partir del análisis de una serie de frames. Ya se han mencionado oportunamente los errores que se acarrean durante todo el proceso, y cómo influyen en la determinación de las coordenadas de un objeto. Por lo tanto, en el momento de realizar la predicción de trayectorias debe considerarse en todo momento que la información de la cual se dispone contiene un cierto grado de error, posiblemente desconocido.

4.4.3.2 Modelo del movimiento

Para poder realizar una buena estimación del recorrido futuro de un objeto es necesario conocer de antemano las características que posee el movimiento. Es muy importante poder modelar el movimiento de un objeto con la mayor precisión posible. Si se dispone de antemano de un buen modelo del movimiento del objeto, será posible realizar mejores predicciones que cuando no se conocen a priori las características del movimiento. Los métodos que se utilicen para la estimación deben tener en cuenta el modelo de movimiento para que la estimación sea exitosa.

Al igual que ocurre en las restantes etapas del sistema, los métodos a utilizar están estrechamente relacionados con las características de un caso en particular del problema. Una solución exitosa bajo ciertas condiciones puede ser un fracaso en la resolución de un problema similar bajo otras condiciones.

A continuación se analizan algunas de las características que puede poseer el movimiento de un móvil, y cómo influye cada una en la predicción de las trayectorias.

4.4.3.3 Características del movimiento. Cómo influyen a una buena predicción.

En el momento de resolver el problema de seguimiento de trayectorias se realizaron ciertas suposiciones acerca de las características del movimiento de un objeto. Se hizo hincapié en la propiedad de consistencia de movimiento, la cual afirma que la trayectoria realizada por un móvil no cambia bruscamente. La dirección y velocidad del movimiento se mantienen relativamente sin cambios si se analizan desplazamientos consecutivos de un objeto.

En base a esas suposiciones se armaron las trayectorias buscando maximizar la suavidad de los recorridos de los objetos.

Las siguientes figuras ilustran dos trayectorias posibles. Resulta evidente que la trayectoria de la izquierda (Gráfico 4-26) tiene una mayor suavidad que la trayectoria de la derecha (Gráfico 4-26)

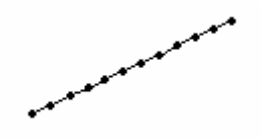


Gráfico 4-26

Gráfico 4-26

Es claro que si el recorrido de un objeto se asemeja al ilustrado en la figura de la izquierda (Gráfico 4-26), en cualquier punto del recorrido puede realizarse una estimación de los movimientos futuros del objeto buscando una trayectoria que se ajuste con la mayor exactitud posible a la información del recorrido previo. En este caso el modelo del movimiento es lineal y la estimación puede realizarse cometiendo un error admisible.

En cambio, en recorridos como el de la figura de la derecha (Gráfico 4-26), donde existen cambios bruscos y totalmente aleatorios, la información previa del movimiento del móvil es poco útil para estimar las posiciones futuras. En tales casos sólo se podrá tener en cuenta la dirección y velocidad actuales para predecir las posiciones futuras a muy corto plazo, ya que los cambios en el movimiento son completamente aleatorios e impredecibles.

De esto se concluye que las estimaciones serán mejores a medida que exista una mayor consistencia en el movimiento. Dicho de otra forma, las estimaciones tendrán mayor exactitud cuanto más se asemejen las trayectorias al modelo del movimiento en el cual se basa la predicción.

4.4.3.4 Información tenida en cuenta para realizar una buena predicción

Analizando las características del movimiento de los objetos, parece ser una conclusión válida que cuanto mayor suavidad tenga el movimiento de un objeto, y más se asemeje al

modelo de movimiento en el cual se basa la estimación, mejores predicciones se podrán realizar sobre su futura trayectoria.

De esta afirmación surgen las siguientes preguntas:

- ¿Cuánta información del recorrido previo es relevante y aporta datos significativos para la estimación del recorrido futuro?
- Si se utiliza más información del recorrido previo, ¿siempre se obtendrán mejores predicciones?
- ¿Cómo influye la frecuencia de muestreo? ¿A mayor frecuencia de muestreo, más datos útiles se poseen?

Resulta claro que si un objeto en movimiento cambia bruscamente su dirección en un punto, gran parte de la información del movimiento previo no aportará datos significativos para la estimación de la trayectoria futura. Por el contrario, posiblemente la estimación será peor si se tiene en cuenta todo el recorrido.

Veamos esto con un ejemplo. Los siguientes gráficos muestran el avance del recorrido realizado por un móvil en tres instantes de tiempo.

Gráfico 4-29

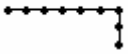


Gráfico 4-29

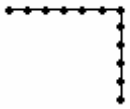


Gráfico 4-29

El móvil comienza realizando un recorrido en línea recta (Gráfico 4-29). En un determinado momento cambia abruptamente su dirección, desviándose en un ángulo recto (Gráfico 4-29). Luego se ve que el móvil mantiene su nueva dirección (Gráfico 4-29).

Supongamos que en el tercer instante de tiempo (Gráfico 4-29) se desea realizar una estimación del futuro recorrido del móvil, basándose en el recorrido previo.

Los primeros puntos del trayecto, antes del giro, no representan información útil para poder estimar la trayectoria futura del móvil. La información que aporta datos relevantes acerca del recorrido futuro es la referente a las posiciones previas inmediatas, más precisamente las que corresponden al trayecto realizado luego del giro.

Veamos otro ejemplo donde el recorrido no tiene cambios bruscos como el ejemplo anterior (Gráfico 4-30).

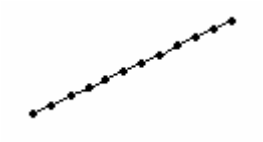


Gráfico 4-30

En este caso el recorrido en su totalidad es lineal. La información de la trayectoria completa contiene información útil para estimar las posiciones futuras.

Del análisis de los dos ejemplos previos surge el siguiente cuestionamiento:

¿Cómo determinar cuáles puntos del recorrido previo contienen información relevante para realizar una buena estimación de la futura trayectoria?

4.4.3.5 Tipo de movimientos de los objetos de la escena

En este trabajo tenemos una escena donde el movimiento de los objetos es rectilíneo uniforme, es decir, los objetos se desplazan en la escena en forma lineal a velocidad constante, pudiendo desaparecer de ella de forma temporal o definitiva.

La predicción de las trayectorias se realiza bajo esas afirmaciones; tenemos por lo tanto un modelo de movimiento lineal. En estas condiciones, el proceso de estimación dará como resultado trayectorias lineales.

4.4.3.6 Descomposición del movimiento en dos componentes

Con el objetivo de modelar el movimiento de un objeto en el plano sobre el cual se desplaza en función del tiempo, lo descomponemos en dos partes. Una parte representa el movimiento del objeto en el eje X del plano, y la otra el movimiento en el eje Y.

De esta forma el movimiento del objeto queda expresado por medio de dos funciones que dependen del tiempo. Llamamos a dichas funciones posX(t) y posY(t). La función posX(t) indica la coordenada en el eje X ocupada por el objeto en el instante de tiempo t, y la función posY(t) la coordenada en el eje Y en el instante t.

Es importante destacar que cuando movimiento de un objeto es rectilíneo uniforme, los desplazamientos en cada uno de los ejes expresados en función del tiempo son funciones lineales.

A continuación se muestra un ejemplo de una trayectoria lineal a velocidad constante realizada por un objeto en siete instantes de tiempo, y su descomposición en posX(t) y posY(t) (Gráfico 4-31, Gráfico 4-32 y Gráfico 4-33).

Nro de frame (t)	**Coordenadas**	PosX(t)	PosY(t)
1	(10, 10)	10	10

2	(15,10)	15	10
3	(20,10)	20	10
4	(25,10)	25	10
5	(30,10)	30	10
6	(35,10)	35	10
7	(40,10)	40	10

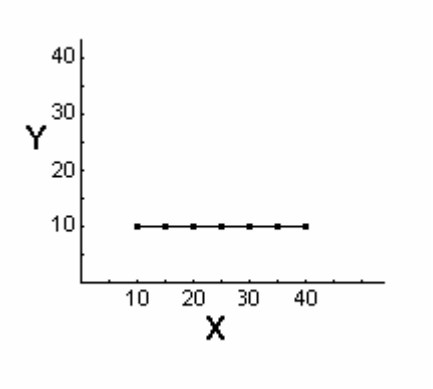


Gráfico 4-31. Movimiento rectilíneo uniforme en 7 frames

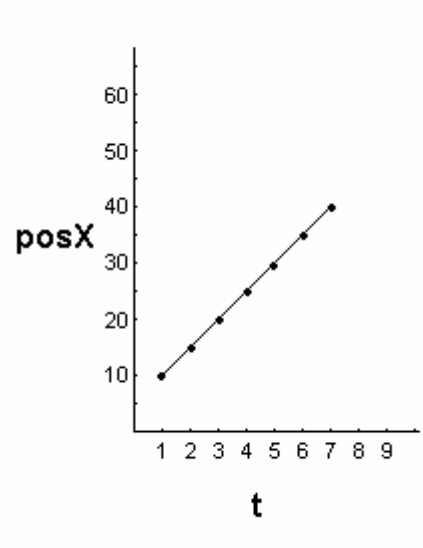


Gráfico 4-32. Ubicación en el eje X en función del tiempo

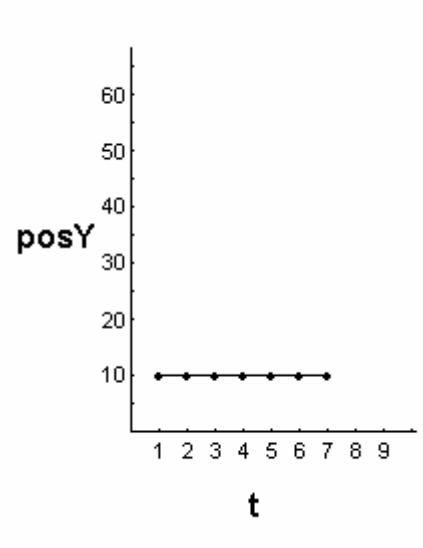


Gráfico 4-33. Ubicación en el eje Y en función del tiempo

4.4.3.7 Estimación utilizando técnica de regresión lineal

Una vez dividido el movimiento del objeto en las componentes posX(t) y posY(t), el problema de predicción de la trayectoria del objeto consiste en estimar los valores que tomarán ambas funciones en instantes de tiempo posteriores.

Si el movimiento del objeto es rectilíneo uniforme, cada uno de los componentes del movimiento expresados en función del tiempo (posX(t) y posY(t)) es una función lineal.

Es importante tener en cuenta el ruido existente en todos los pasos del sistema de seguimiento, por lo que si bien el movimiento real del objeto en cada eje expresado en función del tiempo es lineal, las funciones armadas a partir de las muestras que se tomaron de sus posiciones probablemente no determinen exactamente una recta.

Volviendo al ejemplo presentado previamente, a continuación se ejemplifican posibles valores de las posiciones observadas del objeto en los siete instantes de tiempo considerados (Gráfico 4-34, Gráfico 4-35 y Gráfico 4-36).

Nro de frame (t)	Coordenadas observadas	PosX(t)	PosY(t)
1	(11, 11)	11	11
2	(15,9)	15	9
3	(18,10)	18	10
4	(24,8)	24	8
5	(31,9)	31	9
6	(36,12)	36	12
7	(39,11)	39	11

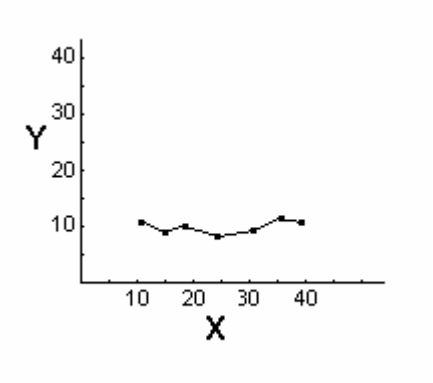


Gráfico 4-34. Muestras del movimiento rectilíneo uniforme en 7 frames

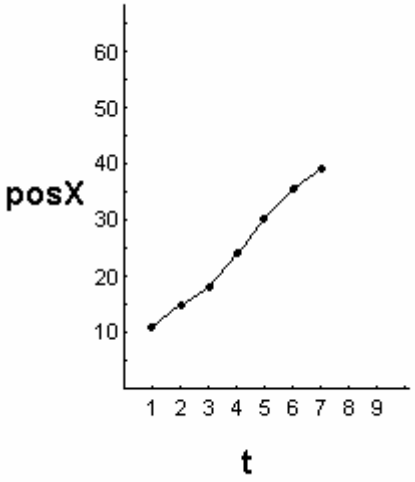


Gráfico 4-35. Ubicación observada en el eje X en función del tiempo

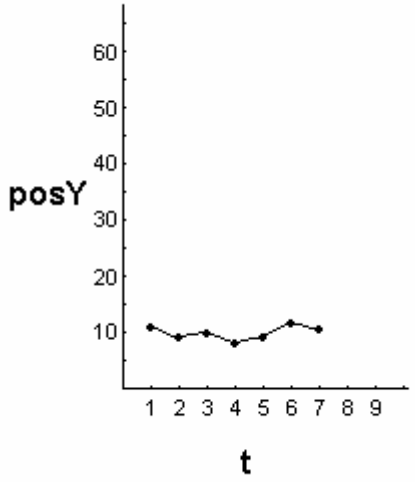


Gráfico 4-36. Ubicación observada en el eje Y en función del tiempo

Nos interesa aproximar la recta que representa el movimiento real del objeto en cada eje, en función de las muestras ruidosas tomadas. La regresión lineal es una buena herramienta para poder realizar la aproximación de PosX(t) y PosY(t).

Las rectas de regresión no solo son útiles para estimar los movimientos reales del objeto a partir de muestras ruidosas, sino que también sirven para estimar las posiciones del objeto en cada eje en instantes posteriores de tiempo.

4.4.3.8 Análisis de regresión

El objetivo de un análisis de regresión es investigar la relación que existe entre una variable dependiente (Y) y una o más variables independientes (X_1, X_2, X_3 ...). Para poder realizar esta investigación, se debe postular una relación funcional entre las variables.

La forma funcional que más se utiliza en la práctica es la relación lineal. Cuando solo existe una variable independiente, esto se reduce a una recta de la forma:

$$Y' = b_0 + b_1 X$$

El símbolo especial Y' es usado para representar el valor de Y calculado por la recta. El valor real de Y rara vez coincide exactamente con el valor calculado, por lo que es importante hacer esta distinción.

Los parámetros b_0 y b_1, representan la ordenada en el origen y la pendiente de la recta. El problema consiste en obtener estimaciones de estos coeficientes a partir de una

muestra de observaciones sobre las variables X e Y. En el análisis de regresión, estas estimaciones se obtienen por medio del método de mínimos cuadrados.

Dada una muestra de n pares (x_i,y_i), las desviaciones e de los valores *y* son:

$e_1 = y_1 – (b_0 + b_1\ x_1)$
$e_2 = y_2 – (b_0 + b_1\ x_2)$
...
...
...
$e_n = y_n – (b_0 + b_1\ x_n)$

Sea $E(b_0,b_1)$ la suma de los cuadrados de todas estas desviaciones, es decir:

$$E(b_0,b_1) = (y_1 - b_0 - b_1 x_1)^2 + (y_2 - b_0 - b_1 x_2)^2 + ... + (y_n - b_0 - b_1 x_n)^2$$

Los valores que minimizan a E(a,b) son aquellos para los que:

$$\partial E/\partial b_0 = 0 \quad y \quad \partial E/\partial b_1 = 0$$

Se obtiene así, un sistema de dos ecuaciones con dos incógnitas b_0 y b_1 cuya solución es:

$$b_1 = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \qquad b_0 = \frac{\sum_{i=1}^{n} y_i - b_1 \sum_{i=1}^{n} x_i}{n}$$

De esta forma se obtienen los parámetros b_0 y b_1 que hacen que la recta se ajuste con el menor error posible a la muestra.

4.4.3.9 Cálculo de las rectas de regresión para el ejemplo planteado

Realizaremos el cálculo de las rectas de regresión que aproximan a las funciones posX y posY para los valores del ejemplo planteado anteriormente. Se utiliza la información de los 7 frames.

En las fórmulas que calculan el valor de b_0 y b_1, los valores x_i corresponden a los instantes de tiempo, mientras que los y_i corresponden al valor de las función que se está estimando (posX o posY).

X_i (tiempo)	Coordenadas observadas	y_i (PosX)	y_i (PosY)
1	(11, 11)	11	11
2	(15,9)	15	9
3	(18,10)	18	10
4	(24,8)	24	8
5	(31,9)	31	9
6	(36,12)	36	12
7	(39,11)	39	11

A continuación se calcula la recta de regresión para posX(t)

$$n = 7$$

$$\sum_{i=1}^{n} x_i\, y_i = 1.11 + 2.15 + 3.18 + 4.24 + 5.31 + 6.36 + 7.39 = 835$$

$$\sum_{i=1}^{n} x_i = 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28$$

$$\sum_{i=1}^{n} y_i = 11 + 15 + 18 + 24 + 31 + 36 + 39 = 174$$

$$\sum_{i=1}^{n} x_i^{\,2} = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 + 7^2 = 140$$

$$b_1 = \frac{n\sum_{i=1}^{n} x_i\, y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^{\,2} - \left(\sum_{i=1}^{n} x_i\right)^2} = \frac{7.835 - 28.174}{7.140 - 28^2} = \frac{973}{196}$$

$$b_0 = \frac{\sum_{i=1}^{n} y_i - b_1 \sum_{i=1}^{n} x_i}{n} = \frac{174 - \frac{973}{196}.28}{7} = \frac{35}{7} = 5$$

Por lo tanto, la recta de regresión posX' que aproxima a la función posX queda armada de la siguiente forma:

$$posX'(t) = 5 + \frac{973}{196}t$$

La siguiente tabla muestra los valores observados y estimados de posX en los 7 instantes de tiempo previos y valores estimados para instantes de tiempo posteriores.

x_i (tiempo)	Coordenadas reales	Coordenadas observadas	PosX (Observado)	PosX (real)	PosX' (aproximado)
1	(10, 10)	(11, 11)	11	10	9.96
2	(15,10)	(15,9)	15	15	14.93
3	(20,10)	(18,10)	18	20	19.89
4	(25,10)	(24,8)	24	25	24.86
5	(30,10)	(31,9)	31	30	29.82
6	(35,10)	(36,12)	36	35	34.79
7	(40,10)	(39,11)	39	40	39.75
8	-	-	-	-	44.71
9	-	-	-	-	49.68
10	-	-	-	-	54.64
11	-	-	-	-	59.61
12	-	-	-	-	64.57

A continuación se calcula la recta de regresión para posY(t)

$n = 7$

$$\sum_{i=1}^{n} x_i y_i = 1.11 + 2.9 + 3.10 + 4.8 + 5.9 + 6.12 + 7.11 = 285$$

$$\sum_{i=1}^{n} x_i = 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28$$

$$\sum_{i=1}^{n} y_i = 11 + 9 + 10 + 8 + 9 + 12 + 11 = 70$$

$$\sum_{i=1}^{n} x_i^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 + 7^2 = 140$$

$$b_1 = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} = \frac{7.285 - 28.70}{7.140 - 28^2} = \frac{35}{196}$$

$$b_0 = \frac{\sum_{i=1}^{n} y_i - b_1 \sum_{i=1}^{n} x_i}{n} = \frac{70 - \frac{35}{196}.28}{7} = \frac{65}{7}$$

Por lo tanto, la recta de regresión posY' que aproxima a la función posY queda armada de la siguiente forma:

$$posY'(t) = \frac{65}{7} + \frac{35}{196}t$$

La siguiente tabla muestra los valores observados y estimados de posX en los 7 instantes de tiempo previos y valores estimados para instantes de tiempo posteriores.

x_i (tiempo)	Coordenadas reales	Coordenadas observadas	PosY (Observado)	PosY (real)	PosY' (aproximado)
1	(10, 10)	(11, 11)	11	10	9.46
2	(15,10)	(15,9)	9	10	9.64
3	(20,10)	(18,10)	10	10	9.82
4	(25,10)	(24,8)	8	10	10
5	(30,10)	(31,9)	9	10	10.18
6	(35,10)	(36,12)	12	10	10.36
7	(40,10)	(39,11)	11	10	10.54
8	-	-	-	-	10.71
9	-	-	-	-	10.89
10	-	-	-	-	11.07
11	-	-	-	-	11.25
12	-	-	-	-	11.43

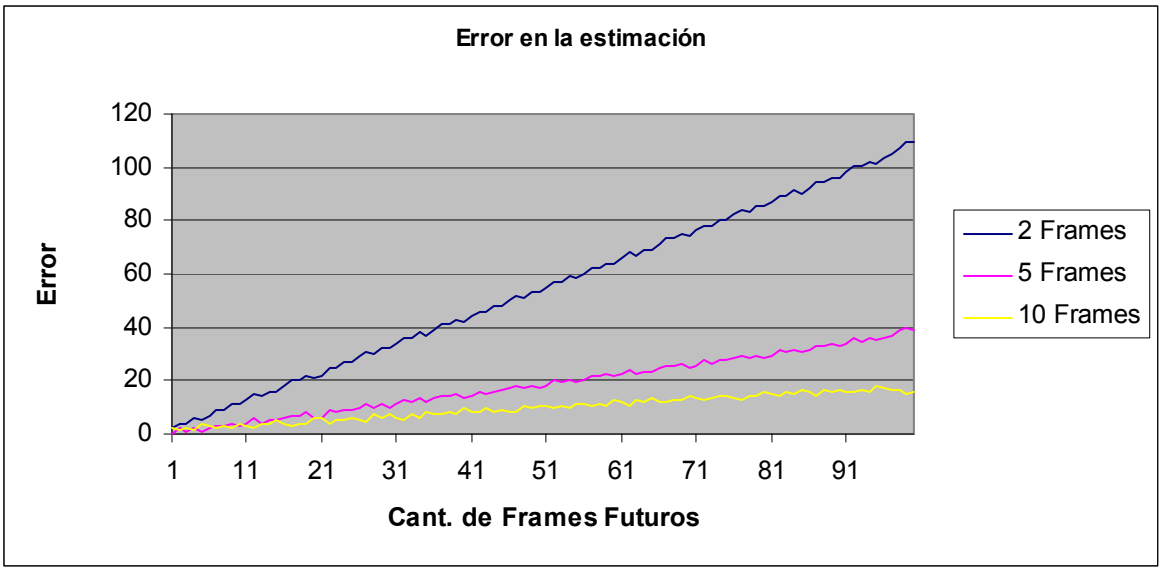
4.4.3.10 Error cometido en la estimación

El error cometido en la estimación de la trayectoria de un objeto se puede calcular como la distancia entre la posición estimada para un instante de tiempo determinado y la posición real del objeto en ese instante de tiempo.

Resulta evidente que el error en la estimación será mayor cuanto más lejano sea el instante de tiempo que se está prediciendo. Otra afirmación acertada es que para el tipo de movimiento analizado el error disminuirá si se utiliza más información del recorrido previo para realizar la predicción.

En esta sección analizamos el error cometido en la estimación del movimiento de un objeto mediante el método utilizado para un conjunto de datos de prueba.

Se analizan estimaciones realizadas mediante regresión lineal utilizando diferente cantidad de información de la trayectoria previa del objeto. El siguiente gráfico ilustra el error cometido en la estimación. La función de error graficada corresponde a un promedio obtenido de un conjunto de pruebas.



El eje X corresponde al instante de tiempo que se estima. El tiempo se determina en función del número de frame futuro. El error se calcula como la distancia en píxeles entre las coordenadas estimadas del objeto y las reales (las cuales obviamente son conocidas cuando llega el instante de tiempo correspondiente). En el gráfico se observan tres curvas de error. Una corresponde a la estimación utilizando los dos últimos puntos observados de la trayectoria del objeto, otra a la estimación utilizando información correspondiente a los cinco instantes de tiempo previos y la última a la estimación utilizando las diez últimas posiciones observadas de la trayectoria.

Se observa que el error cometido disminuye a medida que se utiliza más información para realizar la predicción. Utilizando información de los diez frames previos, el error cometido en la estimación a 50 instantes posteriores de tiempo (frames) es menor a 10 píxeles. En cambio en la estimación que utiliza información de las dos últimas posiciones el error cometido para la predicción del mismo instante de tiempo es mayor a 40.

4.4.3.11 Conclusión

El método de estimación utilizado para predecir la trayectoria de un objeto depende directamente de las características del movimiento y debe basarse en muestras ruidosas del recorrido previo.

Se analizó el tipo más sencillo de movimiento, que es el movimiento rectilíneo uniforme. A partir de las muestras del movimiento realizado por el objeto se lo descompuso en dos funciones que representan el movimiento en cada uno de los ejes. En el tipo de movimiento analizado, dichas funciones determinan una relación lineal. La técnica de regresión lineal da una buena aproximación a las funciones.

En una situación donde el movimiento no es rectilíneo uniforme, la técnica de aproximación debe adaptarse para que se ajuste al modelo del movimiento. Por ejemplo, en el caso que un móvil presente un recorrido circular, el movimiento en cada eje representará funciones senoidales. Puede utilizarse alguna técnica que aproxime las muestras obtenidas del desplazamiento en cada eje a funciones seno.

4.4.3.12 Predicción de colisiones entre trayectorias

La etapa de predicción de las trayectorias genera funciones que permiten estimar para cada una de las trayectorias las coordenadas X e Y del objeto en cada instante de tiempo. Esas funciones permiten extender una trayectoria a k frames posteriores y pueden utilizarse para detectar posibles colisiones entre diferentes trayectorias. El proceso de detección de choques debe considerar que las estimaciones cuentan con error y que el error aumenta a medida que el instante de tiempo que se estima es mayor. Es decir, lo más probable es que la trayectoria estimada difiera cada vez más de la real a medida que el instante de tiempo que se considera es más lejano.

Para realizar la estimación de colisiones se considera un plazo máximo de tiempo *f* (frames a futuro). Las trayectorias estimadas deben extenderse por lo menos a *f* instantes de tiempo posteriores. Otro parámetro importante es la distancia máxima a la cual dos objetos deben estar para considerar una posible colisión entre ellos. Es decir, si llamamos *d* a tal distancia, se considerará que puede existir una colisión entre dos objetos si la distancia entre sus coordenadas estimadas en un frame es inferior a *d*.

A continuación se muestra un algoritmo de predicción de colisiones entre las trayectorias.

```
Entrada:
      f:    cantidad de tiempo futuro en que se estiman las colisiones
      T1..Tm: trayectorias detectadas hasta el frame i prolongadas
mediante estimación a f instantes de tiempo posteriores
      d: distancia máxima en la que dos objetos deben estar para
considerar una colisión

Salida:
      C: conjunto de ternas (Ti,Tj,nFrame) de colisiones estimadas, donde
Ti y Tj son las trayectorias involucradas y nFrame el número de frame
donde se estima que se producirá la colisión

FOR t = i + 1 hasta i + f   // iteración sobre el tiempo
      FOR t1 = 1 hasta m-1   // iteración sobre las trayectorias
            FOR t2 = t1 + 1 hasta m
            // t1(t) y t2(t) son las coordenadas de las trayectorias t1 y
            // t2 en el instante de tiempo t obtenidas con las funciones
            // de estimación posX(t) y posY(t)
                  Si distancia(t1(t), t2(t)) < d
                        agregar(t1,t2,t) a C
```

5 Solución del problema utilizando procesamiento paralelo

5.1 Introducción

En los capítulos anteriores se analizaron cada una de las etapas del sistema y los algoritmos que se pueden aplicar para resolver cada una de ellas.

En problemas del mundo real, cuando las escenas son extensas, suelo ocurrir que hay una fuerte limitante en el tiempo de procesamiento cuando se quieren dar respuestas en tiempo real.

En este capítulo se plantea una solución al sistema mediante el uso de técnicas de programación paralela.

La solución paralela se base en tomar la escena extensa, dividirla en diferentes regiones y procesarlas en paralelo.

5.2 Arquitectura

5.2.1 El sistema adquisidor y la paralelización

Existen básicamente dos formas en las que se puede adquirir imágenes correspondientes a un escenario de interés: utilizando un único adquisidor o utilizando múltiples adquisidores. El primero puede utilizarse cuando se dispone de un adquisidor con capacidad para cubrir toda la escena, el segundo en el caso contrario donde la escena no puede cubrirse por un único adquisidor.

Podemos clasificar el sistema en función de la cantidad de dispositivos de captura y la cantidad de procesadores.

Según la cantidad de dispositivos de captura (C) y procesadores (P):

C=1, P=N: Todos los procesadores trabajan sobre la misma información (Gráfico 5-1).

C=N, P=N: Cada procesador trabaja sobre la información recibida de su propio dispositivo de captura (Gráfico 5-2).

C=M, P=N con M<N: Cada dispositivo de captura provee información a un subconjunto de procesadores.

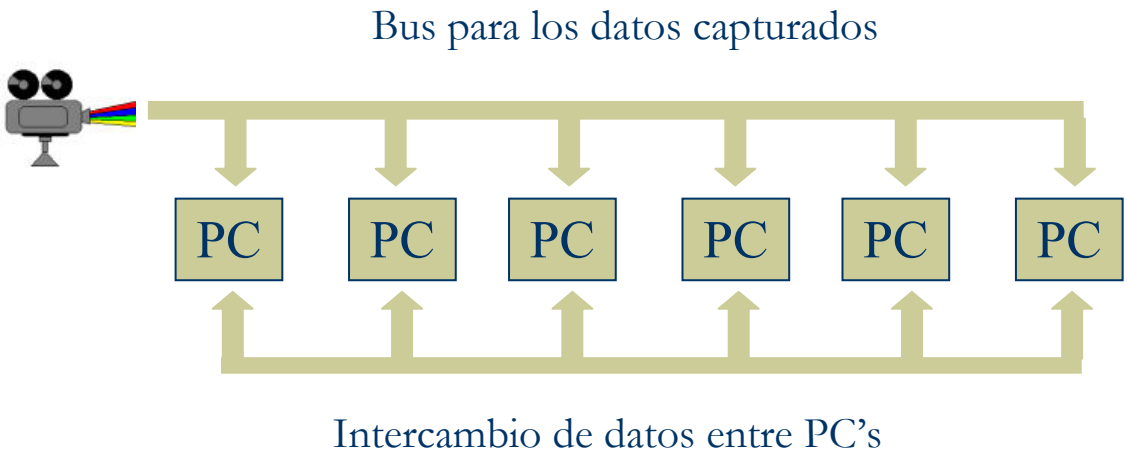


Gráfico 5-1. Dispositivo de captura único, múltiples procesadores

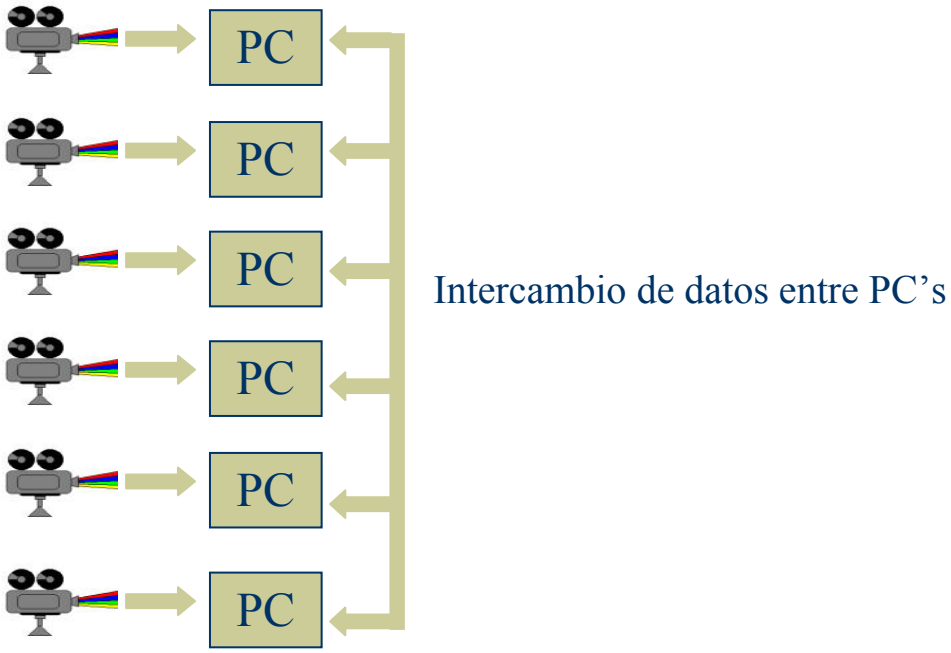


Gráfico 5-2. Un dispositivo de captura por cada procesador.

La solución implementada se adapta a cualquiera de los casos presentados anteriormente.

5.2.2 Paralelización con PVM

PVM (Paralel Virtual Machine) es una herramienta diseñada para solucionar una gran cantidad de problemas asociados con la programación paralela. Para ello, crea una nueva abstracción, que es la máquina paralela virtual, empleando los recursos computacionales libres de todas las máquinas de la red que se pongan a disposición de la biblioteca. Se disponen de todas las ventajas económicas asociadas a la programación distribuida, ya que se emplean los recursos hardware de dicho paradigma; pero programando el conjunto de máquinas como si se tratara de una sola máquina paralela, lo cual es mucho más cómodo.

La máquina paralela virtual es una máquina que no existe, pero un API apropiado permite programar como si existiese. El modelo abstracto que de la PVM consiste en una máquina multiprocesador completamente escalable (es decir, puede aumentarse o disminuirse el número de procesadores en caliente).

5.2.3 Cluster de PC's con PVM

Los algoritmos implementados fueron construidos sobre PVM, lo cual permite abstraerse del hardware y ejecutarse en cualquier plataforma en la que pueda instalarse PVM.

En particular el desarrollo y las pruebas fueron realizados en un cluster de PCs, donde cada una de ellas tiene un único procesador. Para maximizar el uso se utiliza un único proceso por procesador.

5.3 Paralelización de los algoritmos

A continuación se ilustran las etapas y los componentes del sistema como fueron descriptas en el capítulo previo (Gráfico 5-3).

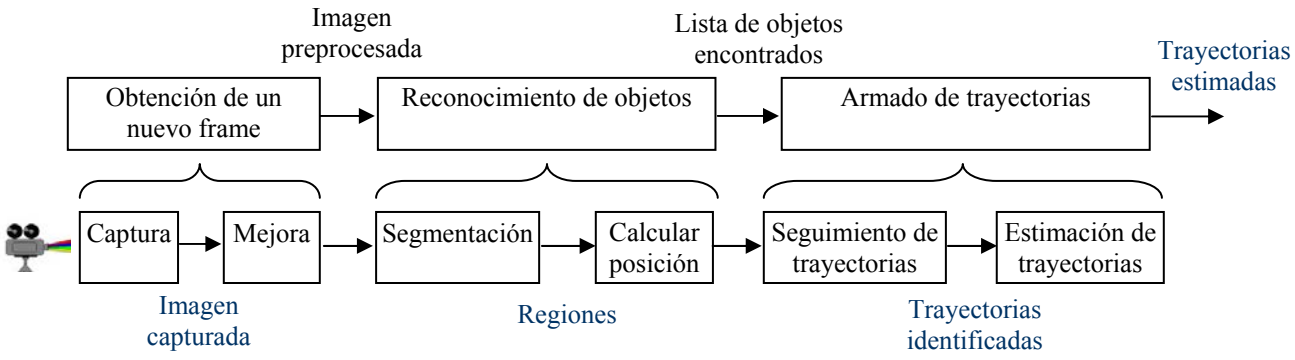


Gráfico 5-3. Etapas en la resolución del problema

Previamente se analizaron los algoritmos que se aplican en cada una de las etapas. Veremos ahora la forma de adaptar cada uno de ellos para ejecutarlos en la arquitectura paralela de forma de aprovechar las ventajas que el procesamiento paralelo ofrece.

5.3.1 Obtención de un nuevo frame para el procesamiento

La obtención de un nuevo frame para el procesamiento abarca dos etapas, la propia adquisición del frame desde la fuente de captura y la aplicación de los algoritmos de mejora con el objetivo de simplificar las etapas posteriores.

En el caso de que el frame sea obtenido mediante múltiples adquisidoras es posible que exista un solapamiento entre las distintas escenas que componen el frame, este solapamiento debería ser conocido a priori por cada proceso para que el mismo pueda manejar la situación y obviar la información redundante. El solapamiento es un dato conocido que se obtiene luego de la etapa de graduación y calibración del sistema adquisidor.

Debido a que en este trabajo se utilizan imágenes artificiales que simulan una escena real y que se encuentran previamente generadas en un archivo con formato de video mpeg, la adquisición del frame correspondiente al instante de tiempo actual consiste en la lectura del archivo de tal frame. Cada nodo del cluster tiene localmente el archivo mpeg correspondiente a la parte de la escena que debe procesar.

5.3.2 Mejora de la imagen

En una situación como la planteada, donde la escena completa es capturada por múltiples cámaras y cada porción de la escena es procesada por un proceso, se puede dar el caso que los algoritmos de mejora que deben aplicarse no sean los mismos para todas las imágenes. Por ejemplo, una cámara puede tomar imágenes que presentan un mayor nivel de ruido que otras, o pueden existir problemas relacionados con el contraste en alguna de las imágenes, etc.

En nuestro caso suponemos que el ruido impulsivo está presente en todas las imágenes. Para eliminarlo debe aplicarse un filtro por la mediana. Por lo tanto cada proceso debe aplicar el filtro por la mediana sobre su porción de la escena con el objetivo de reducir el ruido presente.

5.3.3 Reconocimiento de objetos

5.3.3.1 Segmentación

Por las características del problema y las suposiciones realizadas la segmentación está compuesta por el proceso de umbralización y el etiquetado.

Tanto la umbralización, como el etiquetado se aplican localmente, es decir cada proceso trabaja sobre su porción de la escena. En el caso del etiquetado se necesita un trabajo adicional para resolver el problema de bordes que se expone a continuación.

5.3.3.1.1 Umbralización

El umbral es conocido a priori y es local a cada proceso, ya que es posible que varíe dependiendo del sistema de adquisición.

5.3.3.1.2 Etiquetado

Cada proceso realiza el etiquetado de su porción de escena, pero es necesario agregar un procesamiento adicional y centralizado para resolver las uniones entre los bordes.

Resolver las uniones entre los bordes implica dos condiciones principales:

- Que no existan objetos distintos con la misma etiqueta.
- Que un mismo objeto no esté etiquetado con más de una etiqueta.

Está claro por la forma en que trabaja el algoritmo de etiquetado que dentro de cada región las dos condiciones se cumplen, sin embargo globalmente (la escena completa) se dan situaciones entre bordes como las que se ilustran a continuación (Gráfico 5-4).

0	1	1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	2	0	0
0	0	0	0	0	0	2	2	2	0

0	0	0	0	0	0	1	1	1	0
0	0	0	0	0	0	1	1	1	0
0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	2	2	2	0	1	1	1	1	1
0	2	2	2	0	0	1	1	1	0

0	1	1	1	0	0	1	1	1	0
0	1	1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	0	0	0
0	0	0	1	1	1	1	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

0	1	1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	2	0	0
0	0	0	0	0	0	2	2	2	0
0	0	0	0	0	0	2	2	2	0
0	0	0	0	0	0	2	2	2	0
0	0	0	0	0	2	2	2	2	2
0	0	0	0	0	2	2	2	2	2
0	2	2	2	0	2	2	2	2	2
0	2	2	2	0	0	2	2	2	0
0	2	2	2	0	0	2	2	2	0
0	2	2	2	0	0	2	2	0	0
0	0	2	2	0	0	2	0	0	0
0	0	0	2	2	2	2	0	0	0
0	0	0	0	2	2	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Gráfico 5-4. Etiquetado parcial (izquierda) y etiquetado total (derecha)

Para interpretar lo anterior, supongamos se tienen 3 procesos cada uno de los cuales obtuvo como resultado del etiquetado cada una de las partes que se muestran en la figura de la izquierda respectivamente.

Se puede observar que el primer proceso encontró dos objetos, el segundo también dos objetos y el tercero un objeto, por lo cual existen etiquetas únicas en cada pedazo.

Si se unen los 3 pedazos mencionados, formando la escena completa, se observa que en realidad solo existen dos objetos, lo esperado del etiquetado se muestra en la figura de la derecha, donde para cada objeto hay una única etiqueta distinta. En el ejemplo expuesto no se cumplen las condiciones que se enumeraron, motivo por el cual se procede de la siguiente manera para que el resultado del etiquetado sea el esperado.

El primer problema que mencionamos es que se tienen etiquetas repetidas globalmente (aunque únicas por cada región), esto se soluciona simplemente cuantificando la etiqueta por el número de región, de esta manera en el ejemplo de la figura donde existen las etiquetas: 1, 2, 1, 2, 1 nos quedan 1.1, 1.2, 2.1, 2.2, 3.1 y así las mismas son únicas.

El segundo problema que mencionamos es que un objeto puede estar etiquetado por más de una marca, en la figura del ejemplo las etiquetas 1.2, 2.1, 2.2 y 3.1 pertenecen a un mismo objeto. Para solucionar esto lo que hacemos es analizar los bordes opuestos de las regiones consecutivas, si un objeto está ubicado sobre un borde, etiquetado con una marca A y en el borde opuesto de la región consecutiva también existe un objeto etiquetado con una marca B, significa que la marca A y la B pertenecen al mismo objeto. Para ilustrar esto veamos los bordes consecutivos de la figura (Gráfico 5-4) en las regiones 1 y 2 (Gráfico 5-5) y en las regiones 2 y 3 (Gráfico 5-6).

0	0	0	0	0	0	1.2	1.2	1.2	0

0	0	0	0	0	0	2.1	2.1	2.1	0

Gráfico 5-5. Bordes limítrofes de las regiones 1 y 2

0	2.2	2.2	2.2	0	0	2.1	2.1	2.1	0

0	3.1	3.1	3.1	0	0	3.1	3.1	3.1	0

Gráfico 5-6. Bordes limítrofes de las regiones 2 y 3

En el primer caso (Gráfico 5-5) se puede observar que las etiquetas 1.2 y 2.1 pertenecen al mismo objeto. En el segundo caso (Gráfico 5-6) que las etiquetas 2.1 y 3.1

pertenecen al mismo objeto y también que la 2.2 y la 3.1 pertenecen al mismo objeto, con esto podemos deducir que todas las etiquetas pertenecen al mismo objeto.

El razonamiento anterior se puede formalizar de la siguiente manera:

Usamos *R* para representar la relación binaria "pertenece al mismo objeto", por ejemplo A *R* B significa "La etiqueta A pertenece al mismo objeto que la etiqueta B".

- A *R* A: Por definición cada etiqueta pertenece a un objeto distinto, por lo tanto la misma etiqueta pertenece al mismo objeto => *R* es una relación Reflexiva.
- A *R* B si y solo si B *R* A: Si la etiqueta A pertenece al mismo objeto que la etiqueta B entonces la etiqueta B pertenecerá al mismo objeto que la etiqueta A y viceversa => *R* es una relación Simétrica.
- A *R* B y B *R* C implica A *R* C: Si las etiquetas A y B pertenecen al mismo objeto y las etiquetas B y C pertenecen al mismo objeto necesariamente las etiquetas A y C pertenecerán al mismo objeto => R es una relación transitiva.

Como *R* cumple con las 3 propiedades anteriores, *R* es una relación de equivalencia y el conjunto de elementos que determina *R* (todas las etiquetas pertenecientes al mismo objeto) forman una clase de equivalencia.

Por lo tanto para obtener todas las etiquetas que pertenecen al mismo objeto formamos todas las clases de equivalencia, cada una de las cuales representará un objeto. Por último solo tenemos que reemplazar cada etiqueta por el representante de la clase al que pertenece, de esta manera el resultado cumplirá con las condiciones planteadas.

Para resolver el problema, en la práctica, se procede agregando todas las etiquetas (cada etiqueta se cuantifica por el número de región) a un conjunto ajeno (Ver apéndice "El conjunto ajeno"), luego se analizan los bordes de cada región y por cada detección de que dos etiquetas pertenecen al mismo objeto se le indica al conjunto ajeno que las mismas pertenecen a la misma clase de equivalencia.

Por último cuando ya se crearon todas las clases de equivalencias, cada etiqueta se reemplaza por el representante de su clase y de esa manera cada objeto tendrá una única etiqueta y cada etiqueta pertenecerá a un único objeto.

5.3.3.2 Calcular posición

Para calcular la posición de los objetos es necesario fijar un sistema de coordenados global o absoluto.

Se puede definir que cada porción de la escena identifique sus pixeles relativos al (0,0) local, esta posición corresponde a la (X_i,Y_i) global, donde $X_i=\sum Lx_i$, $Y_i=\sum Ly_i$; Lx_i y Ly_i corresponden a la longitud sobre el eje x e y de la región x_i,y_i respectivamente. El índice de la

sumatoria corresponde para todas las regiones tales que se encuentre a la izquierda y arriba de la región respectivamente.

El siguiente gráfico (Gráfico 5-7) muestra una escena, compuesta por 9 regiones, cada una de 5x5 e indica el valor del punto que corresponde al (0,0) local:

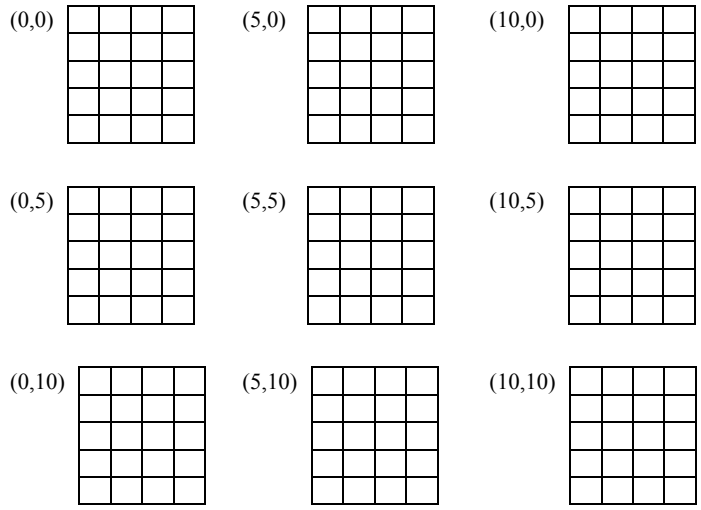


Gráfico 5-7. Coordenadas en una escena dividida en múltiples regiones

Cada proceso por cada etiqueta realiza la sumatoria de las posiciones X e Y (donde X e Y se refieren a las posiciones absolutas) de cada píxel. Luego se centraliza toda la información y por cada etiqueta se realiza la sumatoria de los datos provenientes de los distintos procesos.

Por último se calcula la posición de cada objeto tal como se mencionó en capítulos anteriores.

En la implementación de este trabajo, el sistema de coordenadas es más simple por el hecho de que la escena solo se divide en sectores horizontales.

5.3.4 Armado de trayectorias

Luego de que ya se dispone de todos los puntos que representan cada objeto, el seguimiento y la estimación de la trayectoria se realizan en forma centralizada, por lo que los algoritmos no varían de lo que se explicó en capítulos anteriores.

5.4 Algoritmos implementados

Básicamente la implementación consta de un hilo principal, el cual coordina los demás procesos y realiza las tareas globales, a este lo llamaremos el proceso "principal" o "main".

Luego existen los procesos "procesar_porcion", del cual existirán tantos como la parametrización lo indique y serán los encargados de realizar el trabajo local a cada parte de la imagen (en el caso real, por cada cámara adquisidora de video).

A continuación se muestran los pseudo-código que dan origen a los procesos principal y procesar_porcion. Luego se explica con detalle cada línea.

El siguiente es el pseudo-código del proceso "principal"

```
Entrada:
      nombreVideoMpeg:   El nombre del archivo que contiene el video mpeg se
      va a procesar
      estimarCon: Cantidad de puntos de la trayectoria que se utilizan para
      estimar la trayectoria.
      estimarHasta: Cantidad de puntos de cada trayectoria que se estima.
      cantProcesos: Cantidad de procesos que usará el algoritmo.

Salida:
      Luego de procesado cada frame, se informa los puntos que contiene cada
         trayectoria y los subsiguientes "estimarHasta" puntos estimados

(1) Crear cantProcesos procesos "procesar_porcion"
(2) Iniciar cada "procesar_porcion" con las filas que debe procesar
(3) Para cada frame
      (4) Recibe de cada proceso la cantidad de etiquetas que se encontraron
      (5) Recibe de cada proceso la primer y ultima fila de cada porcion
      etiquetada
      (6) Globaliza las etiquetas
      (7) Envía info a cada proceso para que globalice las etiquetas
      (8) Recibir de cada proceso la posición y la masa de cada objeto
      (9) Unificar la info anterior
      (10) Calcular el centro de masa de todos los objetos
      (11) Agregar puntos a trayectorias
      (12) Estimar las trayectorias
      (13) Mostrar trayectorias y estimaciones
      (14) Esperar el siguiente frame
```

El siguiente es el pseudos-código del proceso "procesar_porcion"

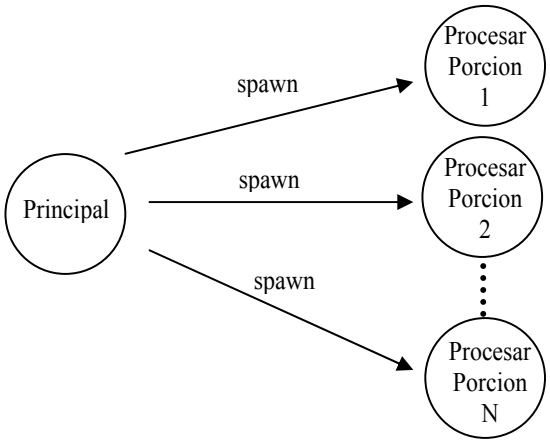
```
(1) Recibe los nros. de fila que debe procesar (desde y hasta)
(2) Para cada frame
      (3) Lee las filas del archivo que le corresponden
      (4) Aplica filtro de la mediana
      (5) Umbraliza
      (6) Etiqueta, con nros. contiguos a partir de 1
      (7) Envia al main la cantidad de etiquetas que encontro
      (8) Envia al main la primer fila etiquetada
      (9) Envia al main la ultima fila etiquetada
      (10) Recibe del main la info para reetiquetar
      (11) Reetiqueta con la info global
      (12) Calcula posiciones y masa de los objetos parciales
      (13) Retorna al main las posiciones y la masa de los objetos
```

5.4.1 Proceso Principal

El proceso principal es encargado de coordinar las actividades que realizan la tarea, es el único que tiene una visión global. Es por ese motivo que se encarga de armar y estimas las trayectorias.

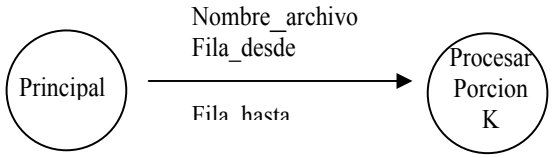
(1) Crear cant_procesos procesos "procesar_porcion"

Crea tantos procesos procesar_porcion como se indica como parámetro. El crear los procesos implica, además de la creación propiamente dicha, que cada uno comience su ejecución.



(2) Iniciar cada "procesar_porcion" con las filas que debe procesar

Es necesario enviar los valores iniciales a cada proceso, en el caso de la implementación vasta con enviar el nombre del archivo a leer y desde que fila hasta cual otra debe procesar. En el caso donde cada proceso procesar_porcion tome los datos de una cámara aquí se deberían indicar los datos propios de cómo acceder a que cámara.



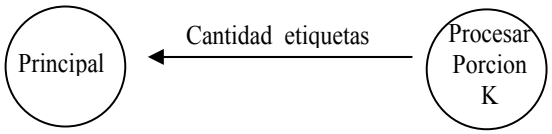
(3) Para cada frame del archivo

Aquí se inicia un bucle donde por cada iteración se procesa un frame. La condición de finalización del bucle viene dada por la circunstancia, en el caso de la implementación particular de la tesis, la condición de fin es haber leído todos los frames del archivo. Sin embargo esta condición podría estar dada por un timer o una interrupción externa.

(4) Recibe de cada proceso la cantidad de etiquetas que se encontraron

Es esta altura cada proceso realizó el trabajo en su porción de imagen correspondiente, por lo que ya se encuentran identificados todos los objetos parciales de cada parte.

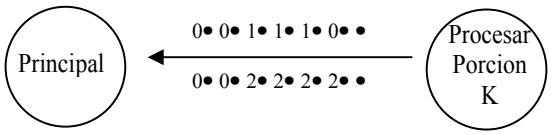
Con recibir la cantidad de etiquetas que se encontraron, el proceso principal ya tiene identificado cuales son, debido a que se etiquetan con números consecutivos a partir del 1. Esta información será necesaria a la hora de calcular cuantos y cuales son los objetos globales.



(5) Recibe de cada proceso la primera y ultima fila de cada porcion etiquetado

Es necesario resolver el problema de bordes ya que hay objetos que pueden haber quedado cortados por dos o más regiones.

Para minimizar el tráfico de los mensajes entre procesos, solo se reciben los bordes de cada región, esta es la información mínima necesaria para conocer si hay objetos inter-regiones.



(6) Globaliza las etiquetas

Aquí es donde se crean las clases de equivalencias explicadas anteriormente.

(7) Envía info a cada proceso para que globalice las etiquetas

Se le envía a cada proceso la información obtenida del paso previo, es decir una asociación entre la marca actual (obtenida del proceso de etiquetado local al proceso) y la marca global. Con estos datos que se envían, cada proceso podrá re-etiquetar su sector con la información global.

(8) Recibir de cada proceso la posición y la masa de cada objeto

Se recibe de cada proceso la información de cada objeto, referente a la masa y la posición de cada uno, tener en cuenta que si bien ya se cuenta con etiquetas globales es posible que los objetos estén cortados, por lo que los datos recibidos en esta etapa hay que mezclarlos.

(9) Unificar la info anterior

Al ya tener las etiquetas globalizadas, unificar la info es tan simple como sumar los valores para cada objeto de todos los procesos.

(10) Calcular el centro de masa de todos los objetos

A esta altura ya se poseen todos los datos necesarios para realizara el cálculo. El cálculo del centro de masa fue explicado en capítulos anteriores, y la implementación es trivial cuando se dispone de toda la información.

(11) Agregar puntos a trayectorias

Luego de haber calculado el punto de masa ya se tiene el punto de referencia de cada objeto, cada uno de estos puntos se agrega a la estructura de datos de las trayectorias usando el algoritmo de seguimiento de trayectoria que se explicó en capítulos anteriores.

(12) Estimar las trayectorias

Para cada trayectoria obtenida se estiman los puntos futuros de los siguientes frames.

(13) Mostrar trayectorias y estimaciones

Se muestran por pantalla todas las trayectorias con todos los puntos estimados de cada una.

5.4.2 Proceso procesar_porcion

Los procesos "procesar_porcion" serán los encargados de procesar las regiones del frame corriente, son los únicos que tienen acceso a los puntos de la imagen.

(1) Recibe los nros. de fila que debe procesar (desde y hasta)

Ya que en la implementación se hizo que los frames sean leídos de un archivo, es necesario inicializar el proceso sabiendo que archivo leer y que porción del mismo deben procesar.

(2) Para cada frame

Aquí se inicia un bucle donde por cada iteración se procesa un frame. La condición de finalización del bucle debe coincidir con la del proceso principal.

(3) Lee las filas del archivo que le corresponden

Lee el siguiente frame del archivo, en particular lo que interesa es la porción del frame que el corresponde al proceso.

(4) Aplica filtro de la mediana

Con el objeto de reducir el ruido introducido en el video se aplica el algoritmo de filtrado por la mediana, tal como se explicó en capítulos anteriores.

(5) Umbraliza

Con el objeto de simplificar la imagen se utiliza la técnica de umbralización, por lo que se aplica el algoritmo de umbralización explicado en capítulos anteriores. El resultado del mismo es una imagen donde cada punto es cero o uno, dependiendo de si es un objeto de interés (1) o fondo (0). Esta técnica es aplicable con un umbral conocido a priori.

(6) Etiqueta, con nros. contiguos a partir de 1

Se aplica el algoritmo de etiquetado explicado en capítulos anteriores y producto de este se obtiene la imagen con los objetos de interés etiquetados con números consecutivos a partir de 1.

(7) Envía al main la cantidad de etiquetas que encontró

Luego de haber finalizado el etiquetado se envía al main la cantidad de objetos que se encontraron, este valor también es útil para identificar las etiquetas ya que el proceso main conoce que las mismas comienzan en 1.

(8) Envía al main la primera fila etiquetada

Con el objeto de resolver el problema de los bordes se envía al main la primera fila, para que este último identifique si se cortó algún objeto entre las distintas regiones.

(9) Envía al mail la ultima fila etiquetada

Con el mismo objetivo que la primera fila se envía la última fila.

(10) Recibe del main la info para reetiquetar

Recibe del main la información necesaria para re-etiquetar los objetos con la información global. La info que recibe es en forma de pares (etiqueta_origen,etiqueta_destino).

(11) Reetiqueta con la info global

Con la información que se recibió en el paso anterior se re-etiquetan las regiones de manera que todas las regiones con la marca etiqueta_origen pasarán a estar marcadas con etiqueta_destino.

(12) Calcula posiciones y masa de los objetos parciales

Recolecta la información parcial a cada región etiquetada para el cálculo de masa.

(13) Retorna al main las posiciones y la masa de los objetos

Retorna al proceso principal la información que obtuvo del paso anterior.

5.4.3 Mejora de la eficiencia de los algoritmos

Es posible realizar una mejora en la eficiencia. Esta mejora economiza el recorrido entero de cada región en cada frame de la película al precio de que el código quede menos natural.

Como se planteó el algoritmo, el etiquetado tiene 3 partes: Etiquetar cada sector, formar las clases de equivalencia y re-etiquetar. Con estos 3 pasos el etiquetado global queda totalmente finalizado, sin embargo la etapa de re-etiquetado puede obviarse y directamente el proceso principal puede utilizar la información de las clases de equivalencias para obtener los centros de masa de los objetos.

El algoritmo del proceso "principal" queda entonces como se muestra en el siguiente pseudo-código:

```
Entrada:
      nombreVideoMpeg:   El nombre del archivo que contiene el video mpeg se
      va a procesar
      estimarCon: Cantidad de puntos de la trayectoria que se utilizan para
      estimar la trayectoria.
      estimarHasta: Cantidad de puntos de cada trayectoria que se estima.
      cantProcesos: Cantidad de procesos que usará el algoritmo.

Salida:
      Luego de procesado cada frame, se informa los puntos que contiene cada
         trayectoria y los subsiguientes "estimarHasta" puntos estimados

(1) Crear cantProcesos procesos "procesar_porcion"
(2) Iniciar cada "procesar_porcion" con las filas que debe procesar
(3) Para cada frame
      (4) Recibe de cada proceso la cantidad de etiquetas que se encontraron
      (5) Recibe de cada proceso la primer y ultima fila de cada porcion
      etiquetada
      (6) Recibir de cada proceso la posición y la masa de cada objeto
      (7) Globaliza las etiquetas
      (8) Unificar la info anterior
      (9) Calcular el centro de masa de todos los objetos
      (10) Agregar puntos a trayectorias
      (11) Estimar las trayectorias
      (12) Mostrar trayectorias y estimaciones
      (13) Esperar el siguiente frame
```

El siguiente es el pseudo-código del proceso "procesar_porcion" modificado

```
(1) Recibe los nros. de fila que debe procesar (desde y hasta)
(2) Para cada frame
      (3) Lee las filas del archivo que le corresponden
      (4) Aplica filtro de la mediana
      (5) Umbraliza
      (6) Etiqueta, con nros. contiguos a partir de 1
      (7) Envia al main la cantidad de etiquetas que encontro
      (8) Envia al main la primer fila etiquetada
      (9) Envia al mail la ultima fila etiquetada
      (10) Calcula posiciones y masa de los objetos parciales
      (11) Retorna al main las posiciones y la masa de los objetos
```

6 Pruebas y conclusiones

6.1 Introducción

En este trabajo se ha realizado una investigación de algoritmos y casos posibles en la resolución de seguimiento y estimación de trayectorias. La misma fue acompañada con una implementación de una solución bajo algunos supuestos.

Este capítulo tiene como objetivo volcar las pruebas realizadas, como así también hacer un análisis de las mismas y sacar conclusiones.

6.2 Infraestructura de ejecución

Para la ejecución de los casos de pruebas se utilizó una máquina virtual (PVM) montada sobre un cluster homogéneo de servidores Linux. El cluster cuenta con 15 nodos y un filesystem compartido de red del cual se levantan los recursos necesarios para la ejecución.

Cada nodo del cluster consta de un procesador Intel Pentium 4 de 2.40GHz, cache de 512 KB y 1GB de memoria RAM.

6.3 Supuestos

- La cantidad de procesadores es igual al número de procesos. Para lograr el mejor rendimiento en la arquitectura paralela se utilizó un único proceso por procesador.
- PVM distribuye un proceso por cada procesador. Efectivamente, el algoritmo de distribución de procesos en procesadores que realiza pvm asegura que si la cantidad de procesos es igual a la cantidad de procesadores, se tendrá un único proceso en cada procesador.

6.4 Pruebas realizadas

Las pruebas ejecutadas se pueden dividir en dos grupos bien diferenciados según el objetivo que persiguen.

- Pruebas de estimación: Orientadas a comprobar el funcionamiento de los algoritmos de estimación implementados.
- Pruebas de rendimiento: Orientadas a medir el tiempo de respuesta en función de la potencia de procesamiento utilizada (cantidad de procesadores).

6.5 Pruebas de estimación

6.5.1 Objetivos:

- Obtener una medida del error cometido en la estimación, en función de la cantidad de información conocida de la trayectoria utilizada.
- Determinar la relación entre el error cometido y la cantidad de frames en el futuro que se estiman.

6.5.2 Que se busca medir

Con esta prueba se intenta obtener una medición empírica del error cometido por el algoritmo de estimación de trayectorias. Se pretende determinar el error en función de la cantidad de información utilizada para la estimación y en función del instante de tiempo futuro que se está estimando.

Para determinar el error cometido se calcula la diferencia entre la posición real observada de un móvil y su posición estimada por el algoritmo.

6.5.3 Conjunto de prueba

Se utilizó un modelo de video con una única trayectoria, debido a que la cantidad de trayectorias no influye en la estimación. El tamaño de video utilizado es de 40.000 píxeles (100x400). El móvil presente en el video tiene una masa de 70 píxeles aproximadamente.

6.5.4 Procedimiento

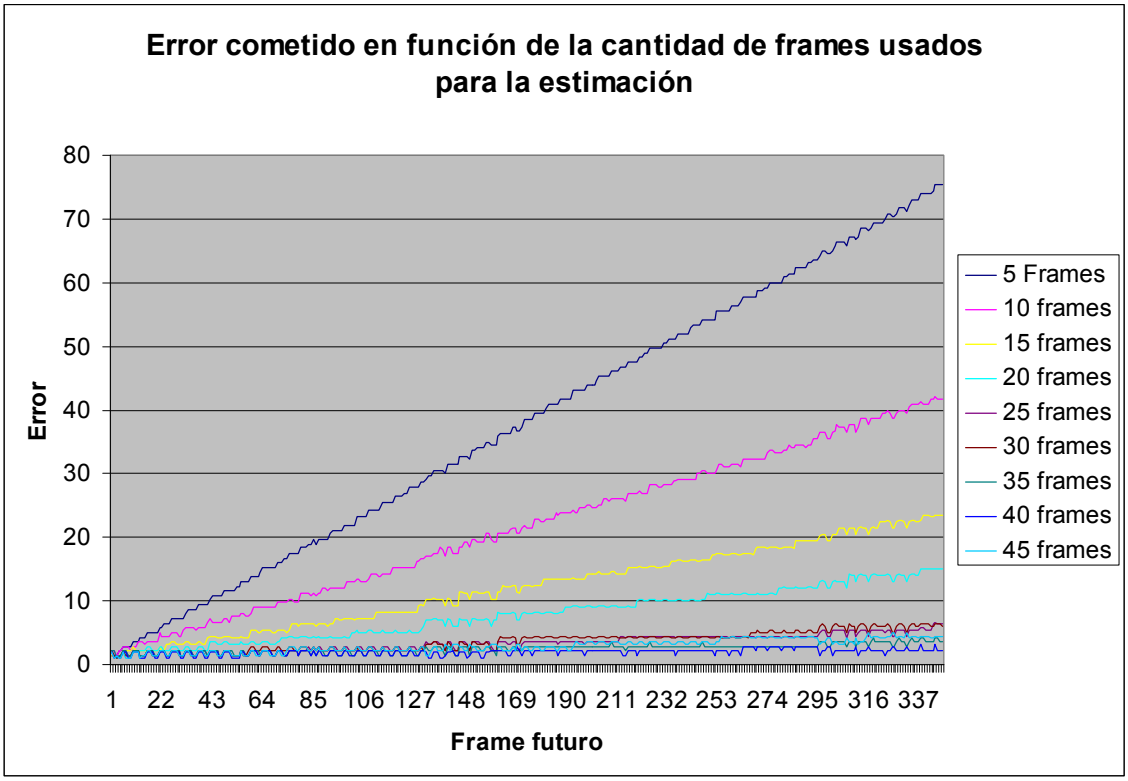
- Colocar el video disponible para ser accedido desde los ejecutables.
- Por cada frame (i:1..N)
    - Obtener posición del objeto (pos_i)
    - Para cada frame previo (j:1..i-1)
        - Recuperar posición que en el frame j se estimó que el objeto iba a estar en el frame i ($posestimada_{ji}$).
        - Calcular error cometido en la estimación realizada en el frame j (diferencia entre pos_i y $posestimada_{ji}$). Acumular el error en diferencias(i-j).
        - Incrementar cantErroresCalculados(i-j)

- Estimar trayectoria futura y almacenarla en ($posestimada_{ik}$) para los frames futuros k que se desean estimar.

- Calcular promedio de error (diferencias(i-j) / cantErroresCalculados(i-j))

6.5.5 Resultados obtenidos

El siguiente gráfico muestra el promedio de error cometido por el algoritmo de estimación. El eje x representa el número de frame futuro que se estima y el eje y el error cometido en píxeles. Las diferentes curvas representan el error cometido variando la cantidad de información utilizada para realizar la estimación.



6.5.6 Conclusiones

Analizando el gráfico se obtienen las siguientes conclusiones:

- Cuanta más información previa se utiliza para estimar los puntos futuros del recorrido de un móvil, más precisa es la estimación.

- Cuanto más cercano es el instante de tiempo que se está estimando, mas precisa es la estimación.
- Para un escenario donde la tolerancia de error es conocida, estas pruebas permitirían determinar la utilidad de la información generada por el algoritmo.

6.6 Prueba de rendimiento

6.6.1 Objetivos

- Determinar como se comportan los algoritmos analizando el tiempo de respuesta bajo distintos escenarios.
- Obtener un umbral que determine el punto óptimo de ejecución en función de la cantidad de procesores.
- Obtener el speedup de los algoritmos.

6.6.2 Descripción del conjunto de pruebas

El conjunto de pruebas fue cuidadosamente estudiado y seleccionando según el objetivo que se persigue.

Se generó un modelo de video, el cual se utilizó como base para generar los archivos a procesar en diferentes tamaños en formato mpeg. El objetivo de tener un único modelo de video instanciado en diferentes tamaños es que los resultados de los tiempos de respuesta obtenidos puedan ser comparables entre sí.

Se construyen videos con los siguientes tamaños:

- 40.000 pixeles (100 x 400)
- 1.000.000 pixeles (500 x 2000)
- 4.000.000 pixeles (1000 x 4000)

Nota: Los tamaños están limitados al formato mpg

La cantidad de frames de los videos que se utlizaron para realizar las pruebas es 200.

6.6.3 Ambiente de prueba

El ambiente es el descrito anteriormente en la sección Infraestructura de ejecución, donde se hizo variar la cantidad de procesares con la cual se construye la máquina paralela pvm entre 1 y 15.

Nota: La cantidad de procesadores está limitado al máximo disponible.

6.6.4 Procedimiento

- Colocar los 3 videos disponibles para ser accedidos desde los ejecutables.
- Por cada uno de los videos:
    - Por cada configuración de cluster (desde 1 a 15 nodos)
        - Tomar el tiempo de comienzo
        - Ejecutar los algorimos
        - Tomar tiempo de finalización
        - Obtener la cantidad de frames que se procesan por segundo.

6.6.5 Resultados obtenidos

Las siguientes tablas muestran los resultados obtenidos en las pruebas realizadas con los tres tamaños de video variando la cantidad de procesadores utilizados. En función de los tiempos de procesamiento obtenidos se calcula la cantidad de frames por segundo procesados y el speedup.

Tamaño de video : 100x400 (40.000 pixeles)

Procesadores	Tiempo (segundos)	fps	Speeup
1	1,159	172,56	1,00
2	0,823	243,01	1,41
3	0,749	267,02	1,55
4	0,684	292,40	1,69
5	0,708	282,49	1,64
6	0,812	246,31	1,43
7	0,900	222,22	1,29
8	0,962	207,90	1,20
9	1,074	186,22	1,08
10	1,381	144,82	0,84
11	1,480	135,14	0,78
12	1,491	134,14	0,78
13	1,768	113,12	0,66

14	1,920	104,17	0,60
15	2,016	99,21	0,57

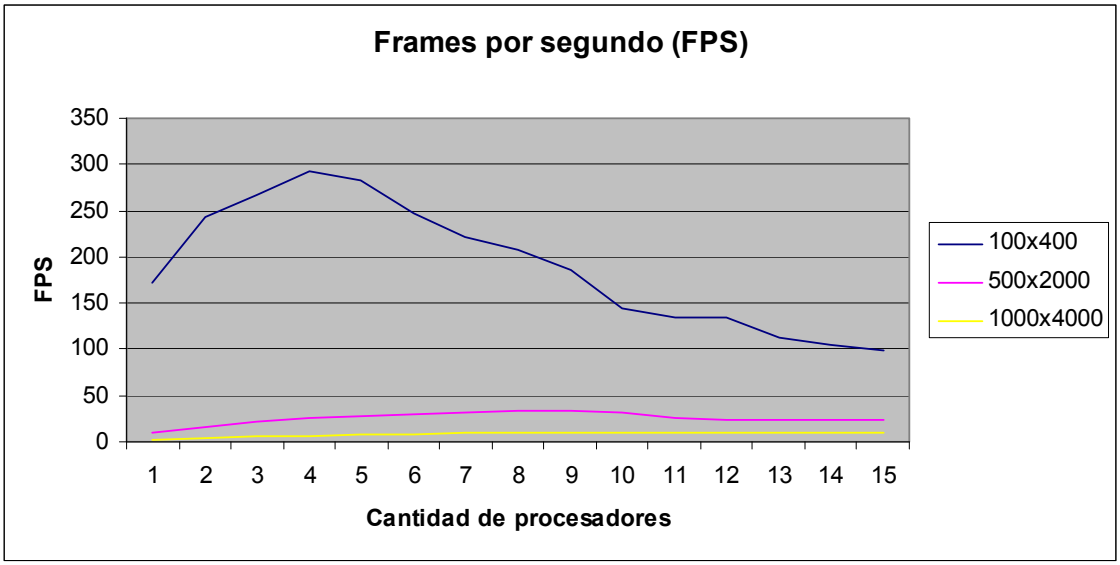
Tamaño de video : 500x2000 (1.000.000 pixeles)

Procesadores	Tiempo (segundos)	fps	Speeup
1	21,680	9,23	1,00
2	12,342	16,20	1,76
3	9,382	21,32	2,31
4	7,784	25,69	2,79
5	6,994	28,60	3,10
6	6,590	30,35	3,29
7	6,248	32,01	3,47
8	5,900	33,90	3,67
9	5,786	34,57	3,75
10	6,238	32,06	3,48
11	7,752	25,80	2,80
12	8,190	24,42	2,65
13	8,250	24,24	2,63
14	8,300	24,10	2,61
15	8,472	23,61	2,56

Tamaño de video : 1000x4000 (4.000.000 pixeles)

Procesadores	Tiempo (segundos)	fps	speeup
1	85,376	2,34	1,00
2	47,640	4,20	1,79
3	34,896	5,73	2,45
4	29,656	6,74	2,88
5	25,956	7,71	3,29
6	22,636	8,84	3,77
7	22,172	9,02	3,85
8	20,868	9,58	4,09
9	20,264	9,87	4,21
10	20,064	9,97	4,26
11	19,840	10,08	4,30
12	19,360	10,33	4,41
13	19,192	10,42	4,45
14	19,200	10,42	4,45
15	19,248	10,39	4,44

El siguiente gráfico resume las pruebas rendimiento.



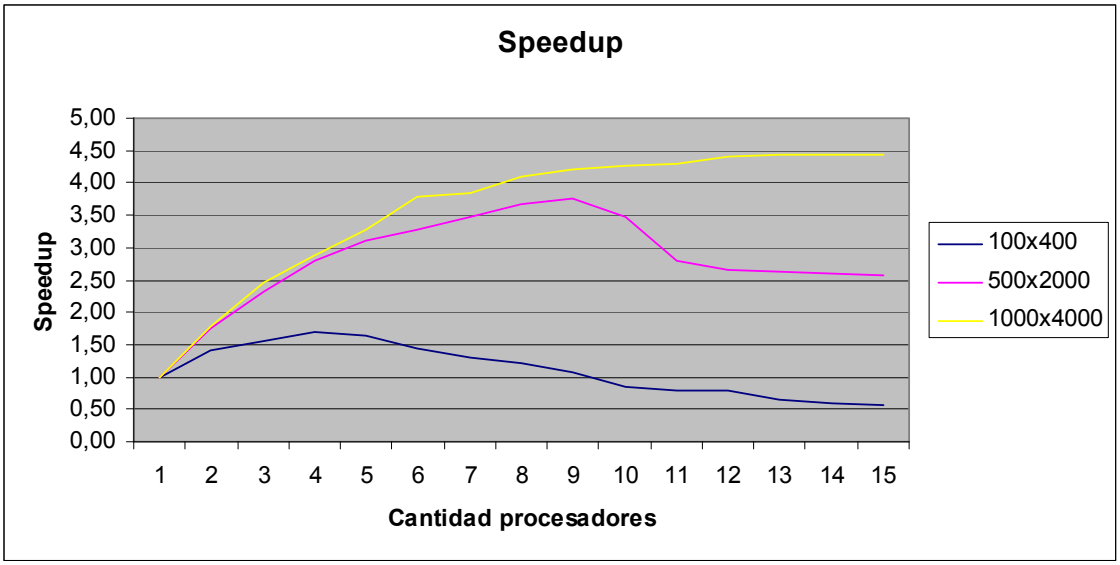
En la figura se muestran 3 curvas, cada una de las cuales representa el comportamiento del poder de procesamiento para cada uno de los videos.

La curva que representa al video mas pequeño (100x400 pixeles) indica que para dicho tamaño de video se pueden procesar muchos mas frames por segundo de lo que requeriría un sistema de tiempo real independientemente de la cantidad de procesadores utilizados. Puede observarse que el máximo procesamiento se logra con 4 procesadores, a partir de allí la curva decrece, indicando que se obtienen peores resultados al agregar más capacidad de procesamiento.

Para el caso del video de tamaño intermedio (500x2000 pixeles) también se observa una suba constante del poder de procesamiento hasta llegar a 9 procesares, lugar donde obtiene su máximo y a partir de aquí decrece, aunque se sigue conservando tiempos de respuesta admisibles para el procesamiento en tiempo real.

Con el video de mayor tamaño (1000x4000 pixeles) se acrecienta lo que pasa con los dos videos anteriores, siendo aquí donde mayores beneficios se obtienen de la arquitectura paralela dado que la curva de procesamiento crece hasta llegar al uso de 13 procesadores, punto donde se estabiliza.

Con la misma información obtenida se realiza el siguiente gráfico de speedup:



Al igual que el anterior se traza una curva por cada video. Cada línea muestra el valor del speedup en función de la cantidad de procesadores. Puede observarse que con el video más pequeño solo se puede alcanzar un speedup de 1,69, mientras que con el intermedio puede llegarse a 3,75 y con el grande se obtiene el mayor beneficio y se llega a 4,45.

6.6.6 Conclusiones

En función de las pruebas realizadas se obtienen las siguientes conclusiones:

- Existe un tamaño de video a partir del cual se hace imprescindible la utilización de un sistema paralelo para lograr tiempos de respuesta admisibles para el procesamiento en tiempo real.
- Existe un punto a partir del cual agregar capacidad de procesamiento no mejora los tiempos de respuesta del sistema. Ese punto es el que determina la capacidad de procesamiento con la cual se logra la eficiencia óptima.
- Con mayores tamaños de video, se obtienen curvas de speedup con mayor crecimiento, donde el punto de inflexión de la curva se presenta en un número de procesadores mas alto. Es decir que la ganancia de utilizar un sistema paralelo es mayor cuanto mayor sea el volumen de datos a procesar.
- Existe una brecha importante entre el speedup óptimo teórico y el práctico, dado que en la práctica no se pudo superar el 4,5 para una arquitectura de 15

nodos. Esto se debe a que el tiempo de comunicación en una arquitectura de pasaje de mensajes en un cluster montado sobre una red ethernet tiene un costo alto. Por este motivo la eficiencia se mejora cuanto mayor es la relación entre volumen de datos a procesar por nodo y la cantidad de comunicación necesaria para resolver el problema.

6.7 Líneas de trabajo futuro

Se dejan abiertas las siguientes líneas de trabajo:

- Realizar estimaciones en escenarios donde los objetos no se mueven bajo un modelo lineal.
- Experimentaciones en clusters formados por nodos con procesadores multicore, combinando procesamiento paralelo con multithreading. Se estima que se lograría mejorar la eficiencia al reducir los tiempos de comunicación entre procesos locales a cada nodo.
- Utilización de imágenes de mayor tamaño. Para las experimentaciones de este trabajo se utilizó el formato de video mpeg-1, el cual limita el tamaño máximo del video.

Apéndice A. Formato Mpeg

A.1. Introducción

Mpeg (del ingles Moving Picture Experts Group) es el nombre de un grupo de trabajo bajo la dirección de la Organización de Estándares Internacionales / Comisión Electrónica Internacional (ISO/IEC), que tiene como objetivo la creación de estándares para video digital y compresión de audio.

Mpeg define la sintaxis del formato de audio y video y también las operaciones que serán emprendidas por los decodificadores.

Los algoritmos empleados por los codificadores no son definidos por mpeg. Este autoriza las continuas mejoras de los codificadores y su adaptación a las aplicaciones específicas que no necesiten alguna redefinición de algún arreglo de los datos.

Además de audio y video codificado, mpeg define métodos que permiten el testeo de la conformidad de estándares de formatos y decodificadores, y publica reportes técnicos.

Mpeg generalmente produce videos de mejor calidad que sus formatos competidores como Video for Windows, Indeo and QuickTime. La mayor diferencia de mpeg comparado con otros formatos de audio y video es que los archivos mpeg son de menor tamaño para la misma calidad

Los archivos mpeg necesitan ser decodificados por hardware especial o por software.

Hay dos principales estándares: Mpeg1 y Mpeg2.

La implementación mas común de mpeg1 estándar provee una resolución de video de 352x240 en 30 frames por segundos, esta calidad de video es levemente inferior a la calidad de los videos convencionales vcr.

Mpeg-2 ofrece resoluciones de 720x480 y 1280x720 en 60 frames por segundo, con una calidad de audio de cd. Esto es superior que la mayoría de las de televisión convencional, incluyendo ntsc y hdtv. Mpeg-2 es usado por dvd-roms y puede comprimir 2 horas de video en unos pocos Gigabytes.

Mpeg, también conocido como mpg, almacena un stream de datos codificados en un medio digital. Es usado para codificar datos de audio, video, texto y gráficos en un único y sincronizado stream.

En la mayoría de los sistemas se usa un hardware especial para la captura de datos mpeg desde un video de la realidad en tiempo real. Cada frame del video capturado es comprimido y almacenado en un stream de datos mpeg. Si el audio también es tomado se codifica y multiplexa en el stream del video con alguna información extra para sincronizar los dos streams juntos para reproducir.

Para reproducir un video en mpeg es necesario disponer un reproductor por hardware o software. El reproductor lee el stream de datos en mpeg, descomprime la información y manda esta al sistema de video y audio del dispositivo.

A.2. Detalles del archivo

Toda la información requerida para reproducir datos en mpeg es codificada directamente en el stream de datos. Por lo tanto no son necesarios encabezados, ni otro tipo de envoltura.

A.3. Compresión en Mpeg

Mpeg usa un método de compresión asincrónico. La compresión sobre mpeg es mucho más complicada que la descompresión, haciendo a mpeg una buena elección para aplicaciones que necesitan escribir los datos una vez pero necesitan ser leídas muchas veces. Mpeg no es una buena elección para los sistemas que requieren que los datos de audio y video sean escritos muchas veces, como un sistema de edición.

Cuando se capturan frames (típicamente 30 frames por segundo para video en tiempo real) habrá muchos datos idénticos en dos o más frames adyacentes. Si un método de compresión toma por sabido esa "redundancia temporal", como muchos métodos de compresión de audio y video lo hacen, entonces no necesitarían codificar el frame entero. En su lugar, solo las diferencias de información entre los frames son codificados. Esto resulta en niveles de alta compresión, con menos datos para ser codificados. Este tipo de codificación es llamada codificación predictiva.

Una reducción en tamaño puede ser logrado por el uso de una predicción bidireccional. La codificación predicativa solo codifica las diferencias entre el frame corriente y el frame previo. La predicción bidireccional codifica el frame corriente basado en las diferencias entre el corriente, previo y próximo frame.

Un stream mpeg contiene 3 diferentes tipos de frames:

- I-frames (intraframe encoded)
- P-frames (predictive encoded)
- B-frames (bi-directional encoded)

Un I-frame contiene un único frame del video y que no depende de la información en ningún otro frame para ser codificado o decodificado. Cada stream de datos Mpeg comienza con un i-frame.

Un p-frame es construido por predicción de diferencia entre el frame corriente y el mas cercano precedente I-frame o P-frame.

Un B-frame es construido desde los dos mas cercanos I-frame o P-frame. El B-frame debe ser posicionado entre I-frames o P-frames.

Una típica secuencia de frames en un stream mpeg puede ser algo como lo siguiente:

IBBPBBPBBPBBIBBPBBPBBPBBI

En teoría, el número de B-frames que puede ocurrir entre dos i-frame o p-frame es ilimitado. En la practica, hay típicamente 12 P-frames y B-frames ocurriendo entre cada I-frame. Un I-frame ocurrirá aproximadamente cada 0.4 segundos de video.

Un stream de datos mpeg no es decodificado y mostrado en el orden en que los frame aparecen en el stream porque los B-frames dependen de dos frames de referencias para la predicción, los dos frames de referencia deben ser decodificados primero.

En el ejemplo previo, el I-frame es decodificado primero, pero antes de los dos B-frames, el P-frame debe ser decodificado y almacenado en memoria junto con el I-frame. Solo luego pueden los dos B-frames ser decodificados desde la información encontrada en los I-frames y P-frames decodificados.

En el ejemplo se asume que estamos situados al comienzo del stream de datos. Los primeros 10 frames son almacenados en la secuencia IBBPBBPBBP (0123456789), pero son decodificados en la siguiente secuencia:

IPBBPBBPBB (0312645978)

Y por último son mostrados en la secuencia:

IBBPBBPBBP (0123456789)

Para la codificación una vez que un I-frame, P-frame o B-frame es construido, es comprimido usando un método de compresión DCT similar a JPEG. La codificación interframe reduce la redundancia temporal (datos idénticos entre frames), la codificación DCT reduce la redundancia espacial (datos correlacionados dentro de un espacio dado). La información de las codificaciones, la temporal y la espacial, es almacenada dentro del stream de datos mpeg.

En la práctica, el tamaño de un P-frames tiende a ser 3 veces inferior a la de un I-frame y el de un B-frame 7 veces inferior.

El estándar de mpeg no obliga a usar P-frames o B-frames. Muchos codificadores evitan la sobrecarga de B-frames o P-frames codificando solo I-frames. Cada frame de video es capturado, comprimido y almacenado entero de forma similar que un mjpeg (motion jpeg). La codificación de los I-frames es muy similar a la codificación de jpeg. Sin la necesidad de las

comparaciones, la codificación puede ejecutarse muy rápidamente: Con un mínimo de asistencia por hadware, la codificación puede hacerse en tiempo real. También el acceso directo al stream de datos es muy rápido porque los i-frames no son tan complejos como los p-frames o b-frames.

Apéndice B. Biblioteca de decodificación de mpeg

En esta sección se explica la librería utilizada en el trabajo para la decodificación de archivos mpeg.

B.1. Introducción

Fue basada en un esfuerzo de la universidad de California con el objeto de tener una librería de decodificación de mpeg totalmente multiplataforma. Surgió de la necesidad de tener una interface simple que permita la extracción de frames desde un archivo en formato mpeg. Fue desarrollada en el instituto neurológico de Montreal en el verano del 1994 para facilitar el desarrollo de un reproductor de mpeg de alta performance para estaciones de trabajo de Silicon Graphics. Sin embargo la librería ha encontrado usos en numerosas aplicaciones.

B.2. Programación con la biblioteca mpeg

Es muy directo su uso, y es similar a como se utiliza tradicionalmente un archivo, se abre para inicializar las estructuras internas, luego se leen los frames hasta que el archivo haya finalizado. Se puede rebobinar en cualquier punto para volver a reproducirlo, pero no se permite el acceso directo a los frames. Cuando se ha terminado se cierra el archivo.

A continuación se muestra un pequeño ejemplo donde se abre un archivo (se recibe como parámetro el nombre) y se leen todos los frames. En el ejemplo se usan funciones como InitializeDisplay() y ShowFrame() que no están implementadas en la librería debido que la visualización de imágenes no es algo que se pueda hacer multiplataforma.

```
#include <stdio.h>
#include "mpeg.h"

int main (int argc, char *argv[])
{
    FILE *mpeg;
    ImageDesc img;
    Boolean moreframes = TRUE;
    char *pixels;
    mpeg = fopen (argv[1], "r");
    SetMPEGOption (MPEG_DITHER, FULL_COLOR_DITHER);
    OpenMPEG (mpeg, &img);
    InitializeDisplay (img.Width, img.Height);
    pixels = (char *) malloc (img.Size);
    while (moreframes)
    {
        moreframes = GetMPEGFrame (pixels);
        DisplayFrame (img.Width, img.Height, pixels)
    }
```

```
    CloseMPEG ();
    fclose (mpeg);
}
```

- Notar que el programa debe abrir y cerrar el archivo que contiene el stream mpeg.
- La estructura ImageDesc tiene toda la información que se puede necesitar para extraer los frames dede el stream.
- SetMpegOption puede ser usado para controlar algunos parámetros de la decodificación.
- No es necesario pasarle parámetros a GetMPEGFrame o CloseMPEG para decirle que mpeg vamos a utilizar ya que depende de variables globales y no es posible decodificar de mas de un mpeg a la vez.

B.3. Conceptos y formatos de datos

En esta sección se tratan conceptos básicos para controlar la librería y mostrar los datos que devuelve. No hace referencia a detalles de cómo se decodifica, codifica o como esta almacenado el mpeg.

B.3.1. Modos de mezcla de colores

"Mezcla de colores" (del ingles dithering) en este contexto es la conversión desde el espacio de color iluminacion-color (aka, YCrCb, YIQ, YUV, que son como están codificados los streams de mpeg y es el mismo espacio usado por las señales NTSC de televisión) hacia alguna forma del espacio RGB.

La experiencia indica que las conversiones para valores de RGB toma mas tiempo y memoria que cualquier otro método, por lo tanto la mayoría de los modos son colores mapeados. Esto significa que OpenMPEG() creará un mapa de colores que puede ser accedido por el puntero ColorMapEntry en ImageDesc, y los valores de píxeles retornados porGetMPEGFrame() son índices a este mapeo de color. El modo de la mezcla de colores afecta la calidad de la imagen decodificada, el número de bits usados por píxel y la profundidad del color de la imagen.

El modo de la mezcla de colores es seleccionado con SeMPEGOption() usando la opción MPEG_DITHER y uno de los siguientes valores:

0	1	2	3	4	5	6	7	8	9
10									
20									
30									
40									
50									
60									
70									79

- ORDERED_DITHER: 8 bits de mapeo de color. Calidad razonable, la decodificación es mas rápida que GRAY_DITHER.
- ORDERED2_DITHER: 8 bits de mapeo de color. Calidad razonable.
- MBORDERED_DITHER: 8 bits de mapeo de color. Calidad razonable.
- FS4_ DITHER: 8 bits de mapeo de color. Colores erróneos.
- FS2_ DITHER: 8 bits de mapeo de color. Colores erróneos.
- FS2FAST_ DITHER: 8 bits de mapeo de color usando la difusión de color Floyd-Steinberg. Calidad razonable.
- HYBRID_ DITHER: 8 bits de mapeo de color. Color pasable.
- HYBRID2_ DITHER: 8 bits de mapeo de color. Levemente peor que HYBRID_ DITHER
- TWOX2_ DITHER: 8 bits de mapeo de color con píxeles duplicados. Calidad pobre.
- GRAY_ DITHER: 256 tonos de grises. Buena calidad y rápida decodificación.
- FULL_COLOR_ DITHER: Alta calidad, 24 bits de color. Lenta decodificación.
- MONO_ DITHER: 1 bit. Monocromo.

8 bits o 24 bits hace referencia a la profundidad de color en la imagen final, es decir el mínimo número de bits alocado para cada píxel. La información final sobre el tamaño del píxel se encuentra en la estructura ImageDesc.

B.3.2. Formato de datos de la imagen

La imagen de datos que retorna GetMPEGFrame() tiene un formato directo. Los píxeles están almacenados por filas ordenadas, comenzando por la esquina superior izquierda. El número de bits alocados por píxel está dado por el campo PixelSize de ImageDesc. Se ilustra una figura que muestra una imagen de 10 x 8 píxeles.

B.4. Referencia de programación

B.4.1. Declaraciones relevantes

```
TypeDef{
    Unsigned char red, green, blue;
}ColormapEntry

TypeDef{
    int Height;  /* En pixeles*/
    int Width;
    int Depth; /* En bits */
    int PixelSize; /* bits por pixel*/
    int Size;
    int BitmapPad;
    int colorMapSize;
    ColormapEntry *Colormap
}ImageDesc;
```

Descripción de los campos:

- `Height`: el alto del video en píxeles.
- `Width`: el ancho de la imagen en píxeles.
- `Depth`: la cantidad de bits por píxel.
- `PixelSize`: el número de bits alocados por píxel.
- `Size`: el tamaño en bytes de un frame sin codificar. Es lo mismo que (`Heigth*Width*PixelSize`)/8 (Notar que se ignora BitmapPad)
- `BitmapPad`: el "quantum" de una línea.
- `ColorMapSize`: el número de entradas en la tabla de mapeo de colores. Usualmente 128. Si se usa un modo que no usa mapeo vale 0.

- `Colormap`: la tabla usada para mapear los valores de los píxeles a valores RGB. Es NULL si usamos un modo que no usa mapeo.

B.4.2. Listado de funciones

`void SetMPEGOption(MPEGOptionEnum Option, int Value);`

Permite setear varias opciones relativas a la decodificación.

`Option` debe tener alguno de los siguientes valores:

MPEG_DITHER
MPEG_LUM_RANGE
MPEG_CR_RANGE
MPEG_CB_RANGE

`boolean OpenMPEG(FILE *MPEGFile, ImageDesc *Image);`

Prepara un stream MPEG para ser decodificado.

`MPEGFile`: El arhivo abierto posicionado al principio.
`Image`: Un puntero a ImageDesc.

`boolean GETMPEGFrame(char *Frame);`

Decodifica el próximo frame de la pelicula. Retorna true si hay algún frame pendiente para decodificar en la película o falso si el frame que decodifico fue el último.

`Frame`: puntero a un sector de memoria alocada. Debe tener suficiente espacio para la imagen decodificada

`void RewindMPEG(File *MPEGfile, ImageDesc *Image);`

Reposiciona el puntero del arhivo MPEG al principio del stream y reinicializa las estructuras las estructuras internas para leerlo nuevamente.

`MPEGfile`: El puntero al stream que fue pasado a OpenMPEG().
`Image`: El descripto de la imagen que fue pasado y llenado por OpenMPEG().

Apéndice C. PVM (Parallel Virtual Machine)

PVM -Parallel Virtual Machine- es una herramienta diseñada para solucionar una gran cantidad de problemas asociados con la programación paralela. Sobre todo, el monetario. Para ello crea una nueva abstracción, que es la máquina paralela virtual, empleando los recursos computacionales libres de todas las máquinas de la red a disposición de la biblioteca. Es decir, se dispone de todas las ventajas económicas asociadas a la programación distribuida, ya que se emplean los recursos hardware de dicho paradigma; pero programando el conjunto de máquinas como si se tratara de una sola máquina paralela, que es mucho más cómodo.

La máquina paralela virtual es una máquina que no existe, pero un API apropiado permite programar como si existiese. El modelo abstracto que permite usar el API de la PVM consiste en una máquina multiprocesador completamente escalable ya que puede aumentarse o disminuirse el número de procesadores en caliente. Para ello, ocultar la red que se esté empleando para conectar las máquinas, así como las máquinas de la red y sus características específicas. Este planteamiento tiene numerosas ventajas respecto a emplear un supercomputador, las más destacadas son:

- **Precio**. Así como es mucho más barato un computador paralelo que el computador tradicional equivalente, un conjunto de ordenadores de mediana o baja potencia es muchísimo más barato que el computador paralelo de potencia equivalente. Al igual que ocurría con el caso del computador paralelo, van a existir factores -fundamentalmente, la lentitud de la red frente a la velocidad del bus del computador paralelo- que van a hacer de que sean necesarios más ordenadores de pequeña potencia que los teóricos para igualar el rendimiento. Sin embargo, aun teniendo esto en cuenta, la solución es mucho más barata. Además, al no ser la PVM una solución que necesite de máquinas dedicadas -es decir, el daemon de PVM corre como un proceso más- pueden emplearse los tiempos muertos de los procesadores de todas las máquinas de la red. Por ello, si ya se dispone de una red Unix montada, el costo de tener un supercomputador paralelo va a ser cero, ya que se dispone de las máquinas y la biblioteca PVM es software libre, por lo que no hay que pagar para usarla.

- **Disponibilidad**. Todo centro de cálculo tiene un mínimo de una docena de máquinas arrumbadas en una esquina, y que nadie sabe qué hacer exactamente ya con ellas. Con esa docena de 486 de hace seis años puede instalarse Linux, PVM y añadirlo al supercomputador paralelo virtual que conforma las máquinas que ya teníamos en red.

- **Tolerancia a fallos**. Si por cualquier razón falla uno de los ordenadores que conforman la PVM y el programa que la usa está razonablemente bien hecho, la aplicación puede seguir funcionando sin problemas. En un caso en el que la aplicación va a estar corriendo durante mucho tiempo, es crítico que la aplicación sea tolerante a fallos. Siempre hay alguna razón por la que alguna máquina puede fallar, y la aplicación debe continuar haciendo los cálculos con aquel hardware que continúe disponible.
- **Heterogeneidad**. Puede crearse una máquina paralela virtual a partir de ordenadores de cualquier tipo. La PVM abstrae la topología de la red, la tecnología de la red, la cantidad de memoria de cada máquina, el tipo de procesador y la forma de almacenar los datos.

C.1. Arquitectura de la PVM

La PVM se compone de dos partes. La primera parte es el daemon, que se llama pvmd. En la versión 3 de la PVM, el nombre es pvmd3. El daemon debe estar funcionando en todas las máquinas que vayan a compartir sus recursos computacionales con la máquina paralela virtual. El daemon pvmd3 es el responsable de la máquina virtual de por sí, es decir, de que se ejecuten los programas para la PVM y de gerenciar los mecanismos de comunicación entre máquinas, la conversión automática de datos y de ocultar la red al programador. Por ello, una vez que la PVM esté en marcha, el paralelismo es independiente de la arquitectura de la máquina, y sólo depende de la arquitectura de la máquina virtual creada por la PVM. Esto evita el problema tener que hacer una rutina de codificación y otra de decodificación por cada arquitectura distinta del sistema.

La segunda parte es la biblioteca de desarrollo. Contiene las funciones para operar con los procesos, transmitir mensajes entre procesadores y alterar las propiedades de la máquina virtual. Toda aplicación se ha de enlazar a la biblioteca para poderse ejecutar después. Hay tres archivos de bibliotecas, la libpvm3.a -biblioteca básica en C-, la libgpvm3.a -biblioteca de tratamiento de grupos- y la libfpvm3.a -biblioteca para Fortran-.

C.2. Uso de la PVM

Conocida la PVM, veamos cómo podemos obtener, instalar, configurar y usar la PVM para poder usar las rutinas desarrolladas en este proyecto.

C.2.1. Filosofía de trabajo de un programa para PVM

Un programa para PVM va a ser un conjunto de tareas que cooperan entre si. Las tareas van a intercambiar información empleando paso de mensajes. La PVM, de forma transparente al programador, oculta las transformaciones de tipos asociadas al paso de

mensajes entre máquinas heterogéneas. Toda tarea de la PVM puede incluir o eliminar máquinas, arrancar o parar otras tareas, mandar datos a otras tareas o sincronizarse con ellas.

Cada tarea en la PVM tiene un número que la identifica unívocamente, es su TID -Task Identification Number-. Es el número al que se mandan los mensajes habitualmente. Sin embargo, no es el único método de referenciar una tarea en la PVM. Muchas aplicaciones paralelas necesitan hacer el mismo conjunto de acciones sobre un conjunto de tareas. Por ello, la PVM incluye una abstracción nueva, el grupo. Un grupo es un conjunto de tareas que pueden ser referidas con el mismo código, el identificador de grupo. Para que una tarea entre o salga de un grupo, basta con que avise que entró o salió del grupo. Esto provee un mecanismo muy cómodo y potente para realizar programas empleando modelos SIMD -Single Instruction, Multiple Data-, en el que se dividen los datos en muchos datos pequeños que sean fáciles de tratar, y después se codifica la operación simple y se replica tantas veces como datos unitarios existan al dividir el problema. Para trabajar con grupos, además de enlazar la biblioteca de la PVM -libpvm3.a- hay que enlazar también la de grupos -libgpvm3.a-.

Habitualmente para arrancar un programa para la PVM, se lanza manualmente desde una de las máquinas del conjunto una tarea madre. La tarea se lanza con el comando spawn desde un monitor de la máquina virtual, que se arranca a su vez con el comando pvm. Esta tarea se encargará de iniciar todas las demás tareas, bien desde su función main -que va a ser la primera en ejecutarse-, bien desde alguna subrutina invocada por ella. Para lanzar nuevas tareas se emplea la función pvm_spawn, que devolverá un código de error, asociado a si pudo o no crearla, y el TID de la nueva tarea.

C.2.2. Activación y desactivación de la PVM

La forma más habitual de activar la PVM es cargando el programa pvm y saliendo de él, lo que dejará activada la PVM. Si se desea apagar la PVM basta con cargar otra vez el programa pvm y emplear el comando halt. Obsérvese que, para que un ordenador sea empleado como parte de la PVM, tiene que estar inscrito en el conjunto de máquinas.

C.2.3. Lenguajes

La PVM puede ser empleada de forma nativa como funciones en C y en C++, y como procedimientos en Fortran. Basta para ello con tomar las cabeceras necesarias y enlazar con la biblioteca adecuada, que viene con la distribución estándar. En el caso de C es libpvm3.a y en el del Fortran libfpvm3.a.

Trabajar con otros lenguajes puede ser un poco más complejo. Si el lenguaje permite incorporar funciones nativas en lenguaje C -como es el caso, por ejemplo, de Java- no hay ningún problema, ya que puede invocarse la función; bien directamente si el lenguaje lo permite, bien haciendo alguna pequeña rutina para adaptar el tipo de los datos, el formato de

llamada a función o cualquiera de las restricciones del lenguaje empleado para invocar funciones en C.

C.2.4. Cómo obtener la PVM

La vía más cómoda para obtener la PVM es a través de netlib, en la url http://www.netlib.org/pvm3/index.html.

C.2.5. Manejo del programa pvm

El programa pvm va a corresponder al núcleo de control de la máquina virtual, o, dicho de otra forma, al monitor de la máquina. A todos los efectos, se va a comportar como una consola dentro de la máquina virtual; por eso es denominado muchas veces, simplemente como consola. Cuando es ejecutado, se entra en el intérprete de comandos de nuestra máquina virtual. Él lanza el daemon de la PVM si no se ha lanzado previamente y además permite ejecutar las funciones básicas de control de la PVM. Si invocamos el programa pvm de la forma pvm archivo, el archivo indicado será interpretado como la lista de máquinas que será incorporada a la PVM.

C.2.6. El TID

El TID -Task Identifier- es el identificador de tarea, y se comporta como el PID de un sistema Unix, sólo que de la máquina virtual. Está definido como un entero de 32 bits para aumentar el rendimiento al máximo y al mismo tiempo permitir direccionar el mayor número de tareas posibles.

C.2.7. Las clases de arquitectura

Para evitar el engorro de andar realizando transformaciones continuas de datos, la PVM define clases de arquitecturas. Antes de mandar un dato a otra máquina comprueba su clase de arquitectura. Si es la misma, no necesita convertir los datos, con lo que se tiene un gran incremento en el rendimiento. En caso que sean distintas las clases de arquitectura se emplea el protocolo XDR para codificar el mensaje.

Las clases de arquitectura están mapeadas en números de codificación de datos, que son los que realmente se transmiten y, por lo tanto, los que realmente determinan si es necesario realizar una conversión.

C.3. El modelo de paso de mensajes de la PVM

C.3.1. Modelo de paso de mensajes en PVM

El modelo de paso de mensajes es transparente a la arquitectura para el programador, por la comprobación de las clases de arquitectura y la posterior codificación con XDR de no coincidir las arquitecturas. Los mensajes son etiquetados al ser enviados con un número entero

definido por el usuario, y pueden ser seleccionados por el receptor tanto por dirección de origen como por el valor de la etiqueta.

El envío de mensajes no es bloqueante. Esto quiere decir que el que envía el mensaje no tiene que esperar a que el mensaje llegue, sino que sólamente espera a que el mensaje sea puesto en la cola de mensajes. La cola de mensajes, además, asegura que los mensajes de una misma tarea llegarán en orden entre si. Esto no es trivial, ya que empleando UDP puede que ocurrir que se envíen dos mensajes y que lleguen fuera de orden, ya que UDP es un protocolo no orientado a conexión. TCP, por ser un protocolo orientado a la conexión, realiza una reordenación de los mensajes antes de pasarlos a la capa superior; sin embargo, establecer las conexiones entre nodos empleando TCP supone, si tenemos n nodos, tener un mínimo de (**n)(n-1)** conexiones TCP activas. Vemos que hasta para números pequeños de **n** nos quedamos sin puertos por éste planteamiento. Establecer conexiones TCP entre procesos en lugar de entre nodos es peor todavía, por las mismas razones que en el caso de los nodos.

La comunicación de las tareas con el daemon sí se hace, en cambio, empleando TCP. Esto se debe a que, al ser comunicaciones locales, la carga derivada de la apertura y cierre de un canal es muy pequeña. Además, no se tienen tantas conexiones como en el caso de la conexión entre daemons, ya que las tareas no se conectan entre sí ni con nada fuera del nodo, por lo que sólo hablan directamente con su daemon. Esto determina que serán **n** conexiones TCP, que sí es una cifra razonable.

La recepción de los mensajes puede hacerse mediante primitivas bloqueantes, no bloqueantes o con un tiempo máximo de espera. La PVM permite realizar los tres tipos de recepción.

C.3.2. Los eventos asíncronos

Los eventos asíncronos corresponden al equivalente de las señales del sistema en Linux. Se producen cuando acontece algo inaudito, y puede pedirse al sistema que nos interrumpa independientemente de lo que estemos haciendo para notificarnos de que algo anormal está aconteciendo. Los eventos pueden viajar de una máquina a otra, para informar a todas aquellas que lo precisen. Estos eventos corresponden a cosas tan traumáticas como una tarea que termina en forma anormal, una máquina es eliminada, o la incorporación de máquinas.

C.3.3. Descriptores de mensaje de los eventos asíncronos

Los distintos descriptores de mensaje de los eventos asíncronos son:

- PvmTaskExit: Una tarea acaba o aborta. Una tarea colgada no puede ser detectada, por lo que si una tarea se cuelga no se generará un mensaje.

- PvmHostDelete: Una máquina es eliminada del conjunto de máquinas o se apaga.
- PvmHostAdd: Se incorpora una máquina nueva al conjunto de máquinas.

C.4. Funcionamiento de la PVM

C.4.1. El daemon de PVM con detalle

El daemon de PVM puede ser arrancado bien por un monitor local, bien por un monitor remoto. Lo normal es que en el caso del monitor local sea arrancado como maestro y en el caso del remoto sea arrancado como esclavo, pero esto puede ser modificado en la propia forma de invocación. Después de arrancar, crea los sockets que necesita para hablar con los otros daemons y abre un archivo de históricos de error en /tmp/pvml.uid, donde uid es el identificador de usuario propietario de la PVM.

Cuando el daemon de la PVM ve que su máquina es desconectada del conjunto de máquinas, mata todas las tareas conectadas a él mediante una señal SIGTERM. Después manda un mensaje a la tabla de máquinas para que elimine su entrada. En caso de salidas más salvajes (matar con el comando kill el daemon) la PVM demorará un poco más en descubrir que un nodo no está funcionando, pero acabará descubriéndolo y eliminando el nodo de la tabla de máquinas.

C.4.2. Arranque de los daemon esclavos

El arranque es realizado por el shadow pvmd, que no es mas que un fork() que realiza la pvmd para poder seguir atendiendo mensajes mientas que el shadow pvmd, el proceso hijo, se encarga de la creación de nuevos nodos, proceso bastante más lento. Se puede hacer mediante rsh o mediante rexec. La elección de la forma de arranque será bastante problemática.

rsh es un programa que permite lanzar tareas remotas sin autentificación de usuario. Para que funcione, hay que tener actualizados en todas las máquinas el archivo /etc/hosts.equiv para que todo el cluster aparezca en él, o incluir las máquinas en el archivo .rhosts.

C.5. Comandos de la PVM

C.5.1. Comandos fundamentales del programa PVM

El programa PVM corresponde al intérprete de comandos de la máquina virtual. Algunos de los comandos más importantes son:

add máquina: incorpora la máquina indicada a la máquina paralela virtual.

delete máquina: elimina la máquina indicada del conjunto de máquinas asociadas a la máquina paralela virtual.

conf: configuración actual de la máquina paralela virtual.

ps: listado de procesos de la máquina paralela virtual. ps -a lista todos los procesos.

halt: apaga la máquina paralela virtual. Esto significa que mata todas las tareas de la PVM, elimina el daemon de forma ordenada y sale del programa pvm. Es la forma de salir en condiciones, y mucho mejor que matar el daemon con kill.

help: lista los comandos del programa.

id: imprime el TID de la consola.

jobs: genera un listado de los trabajos en ejecución.

kill: mata un proceso de la PVM.

mstat: muestra el estado de una máquina de las pertenecientes a la PVM.

pstat: muestra el estado de un proceso de los pertenecientes a la PVM.

reset: inicializa la máquina. Eso supone matar todos los procesos de la PVM salvo los programas monitores en ejecución, limpiar las colas de mensajes y las tablas internas y pasar a modo de espera todos los servidores.

setenv: lista todas las variables de entorno del sistema.

sig señal tarea: manda una señal a una tarea.

spawn: arranca una aplicación bajo PVM. Es un comando bastante complejo cuyas opciones veremos en una sección aparte.

trace: actualiza o visualiza la máscara de eventos traceados.

alias: define un alias predefinido, es decir, un atajo para teclear un comando.

unalias: elimina un alias predefinido.

version: imprime la versión usada de la PVM.

Cualquiera de estos comandos, o cualquier combinación de ellos puede ser definida en el archivo .pvmrc; dicho archivo será leído y ejecutado al arrancar cada monitor.

Los monitores pueden ser arrancados en cualquier momento, en cualquier número, y en cualquier máquina asociada a una máquina paralela virtual determinada.

C.5.2. Modificadores spawn

El comando spawn permite lanzar aplicaciones para PVM. Algunas de los modificadores que emplea son:

-count: número de tareas involucradas. Por defecto es 1.

-host: especifica la máquina para lanzar. Por defecto es any.

-ARCH: lanza en máquinas de tipo ARCH.

-?: Activa el modo depuración.

-: Redirecciona la salida de la tarea que se lanzará al monitor.

-archivo: Redirecciona la salida de la tarea que se lanzará al archivo indicado.

-@: Traza el trabajo, redirigiendo la salida al monitor.

-@file: Traza el trabajo, redirigiendo la salida al archivo indicado.

C.5.3. Opciones del archivo de máquinas

El archivo donde se especifican las máquinas es de extrema importancia, ya que define la arquitectura de nuestra máquina paralela virtual. Se compone de una lista de máquinas, que pueden llevar o no el símbolo & al principio según no sean o sean inicializadas al arrancar el monitor, respectivamente. Las líneas de comentario son aquellas que comienzan con el símbolo #. Además de esto, cada máquina puede llevar asociada una serie de parámetros para determinar su configuración. Los más importantes son:

lo=usuario: permite identificar un nombre de usuario distinto para la conexión a la máquina indicada. Por defecto, se emplea el nombre del usuario que lanzó el proceso en la máquina actual.

so=clave: especifica la clave de la máquina remota. No es una buena idea emplear este campo, salvo que la red sea privada. Por defecto intenta una conexión con rsh, que sólo funcionará si en la máquina remota está la máquina local como una entrada al archivo .rhosts y las RPC están activadas.

dx=ruta: donde se encuentra el pvmd3

ep=rutas: conjunto de rutas, separadas por ;, donde se van a encontrar los ejecutables que se van a lanzar en la máquina. Si no se especifica, buscará en el directorio $HOME/pvm3/bin/PVM_ARCH.

sp=valor: valor de potencia computacional relativa respecto a las otras máquinas. Los rangos posibles están entre 1 y 1000000. Es importante tener en cuenta la carga habitual de una máquina. Puede que una máquina de baja potencia sin carga tenga un valor más alto que una máquina de alta potencia con una carga excesiva. El valor por defecto es 1000.

bx=ruta: localización del depurador. Sólo es activado si también se activa la opción de depuración.

wd=ruta: directorio de trabajo. Es, pues, donde las tareas lanzadas se van a ejecutar. Por defecto apunta a $HOME.

ip=dirección: especifica la dirección IP en la que se va a buscar la máquina.

so=ms: significa que el servidor esclavo no va a ser lanzado con rsh, sino de forma manual. Sólo es útil si no se dispone de RPC o no quiere utilizarse por razones de seguridad.

Un parámetro por defecto puede ser redefinido con una línea que comienze por *. En ella se definen los parámetros como si el * fuese una máquina, y los parámetros definidos de esa forma serán los parámetros por defecto hasta nueva redefinición.

C.6. Programación bajo la PVM

Aunque en principio cómo esté codificado un programa que emplee la PVM es algo muy personal de su programador, la estructura básica de un programa para PVM puede ser organizada en un conjunto de bloques estructurales que realizan determinadas funciones básicas. Veamos dichos bloques estructurales.

C.6.1. Esquema de la aplicación bajo PVM para el maestro

El maestro se encarga en un programa común de PVM tanto de un arranque para que la aplicación se ejecute correctamente como de un cierre ordenado de la aplicación. Es útil también para grabar los resultados finales de la aplicación en un archivo y para leer los datos de entrada, si estos tienen que ser difundidos a varios nodos o leídos con una estructura determinada, por lo que resultaría caótico que los leyeran los nodos independientemente entre sí. Un esquema organizado del arranque del maestro puede ser:

- Solicita su TID.
- Se inscribe en un grupo.
- Lanza los esclavos a los nodos que considere oportuno.
- Hace un bloqueo de barrera, esperando a que todos los esclavos se den de alta en el grupo.
- Después de todo esto, ya puede comenzar el bucle principal del maestro.

C.6.2. Esquema de la aplicación bajo PVM para el esclavo

El arranque de un esclavo es bastante más sencillo. Las funciones básicas que un esclavo va a realizar para arrancar son:

- Solicita su TID
- Se da de alta en el grupo de su maestro, y realiza un bloqueo de barrera, esperando que todos los otros esclavos se den de alta.
- Después de esto, comenzará el bucle interno del esclavo.

C.6.3. Solicitud de TID

Antes de ejecutar cualquier instrucción relacionada con la PVM, un proceso debe solicitar un TID. Además de ser importante el TID para gran cantidad de operaciones, la operación de solicitud de TID automáticamente da de alta nuestro proceso para el daemon de

PVM local. En la PVM para Linux no es obligatorio solicitar el TID, ya que toda llamada a la PVM, si el proceso no está dado de alta, lo da de alta automáticamente. Sin embargo, es recomendable, ya que determinadas implementaciones de la PVM -como es el caso de la PVMe para SP/2 bajo AIX- exigen que se solicite el TID antes de comenzar a trabajar. La sentencia es:

```
int tid=pvm_mtyd()
```

Es importante destacar que cumple la función de solicitar un TID sólo si es la primera función de la PVM que invocamos. En otro caso, todo lo que hará será devolver el TID que ya teníamos asignado.

C.6.4. Inscribirse en un grupo de tareas

Inscribir una tarea en un grupo de tareas es una cosa realmente útil, sobre todo cuando desea emplearse la sincronización de barrera.

Para inscribirse en un grupo de tareas, basta que el proceso ejecute la instrucción pvm_joingroup. Esta instrucción tiene la sintaxis:

```
int inum = pvm_joingroup(char *grupo)
```

inum será el identificador de la tarea dentro de un grupo. Siempre puede consultarse por el TID para un indentificador y grupo, o puede preguntarse por el identificador para un TID y un grupo.

Un grupo puede abandonarse con:

```
int info = pvm_lvgroup(char *grupo)
```

C.6.5. Lanzar varias tareas esclavas

Para lanzar varias tareas esclavas, basta con hacer un spawn de las tareas. Podemos tener el caso de varias tareas distintas, en cuyo caso el esquema será:

```
int ntareas=pvm_spawn(char *tarea,char **argumentos,
                int flag,char *host,int ntareas,int *TIDs);
```

repitiendo la línea tantas veces como tareas distintas se deseen mandar, o tareas iguales en máquinas distintas, donde:

char *tarea es el nombre del archivo a ejecutar.

char **argumentos será una matriz con los argumentos.
int flag opciones para lanzar una tarea, indicadas más adelante.
char *host es el nombre de la máquina en la que se va a ejecutar la tarea.
int ntareas es el número de tareas iguales que serán lanzadas en el nodo indicado.
int *TIDs devolverá un arreglo con los TIDs de las tareas creadas.
El valor devuelto por la función será el número de tareas lanzadas con éxito.

Es importante comprobar el número de tareas lanzadas devuelto por la función de spawn. No siempre pueden inicializarse todas las tareas; de hecho, puede ocurrir que no pueda iniciarse ninguna por cualquier problema existente.

C.6.6. Envío de datos

El envío y la recepción de datos son la piedra angular del modelo de la PVM. Para enviar datos, se emplean tres pasos:

- Inicialización del buffer
- Empaquetado del dato
- Enviar el dato

Recordamos que los datos van codificados con XDR, y que, por ello, es preciso empaquetar el dato para asegurar una transmisión correcta.

Cuando comienza a ejecutarse un programa en PVM, existen un buffer activo de emisión y otro de recepción que serán empleados para comunicarse con otros nodos. Para crear algún buffer adicional, se emplea la función:

int bufid = pvm_mkbuf(int decod)

Donde bufid es el identificador de buffer y decod la forma de codificación. Habitualmente no es preciso crear buffers adicionales, por lo que muy raramente se emplea esta instrucción.

Los buffers pueden ser eliminados con la instrucción:

int info = pvm_freebuf(int bufid)

Donde bufid es el identificador del buffer que se va a eliminar.

Otra forma bastante más cómoda para mandar datos es usar la instrucción pvm_psend(), que permite mandar una matriz contigua de datos del mismo tipo. De cualquier forma, esta función tiene gran cantidad de limitaciones.

C.6.7. Inicialización del buffer

Siempre que se vaya a empaquetar un mensaje, hay que ejecutar la instrucción pvm_initsend(), que inicializará el buffer activo. En caso que no hacerlo, los datos que se manden serán los empaquetados más los empaquetados previamente en el buffer. La sintaxis es:

int bufid = pvm_initsend(int encod)

El identificador de buffer devuelto por la función será el del buffer inicializado (el activo). El parámetro que se pasa a la función es la codificación, de definición similar a la del mismo parámetro en la instrucción de creación de un buffer nuevo.

C.6.8. Empaquetado de datos

En la PVM los datos no se pueden mandar tal y como están. Es preciso empaquetar los datos en un buffer y después mandar el buffer. Aquí tenemos una gran diferencia entre C y Fortran. Fortran tiene una instrucción para empaquetar, mientras que C tiene catorce distintas. El formato en once de ellas es:

int info = pvm_pkXXX(YYY *que, int cuantos, int desde_donde)

donde XXX es el tipo en PVM de lo que se va a mandar (byte, cplx, long, etc.), e YYY el tipo en C relacionado. Para empaquetar una cadena no se indica ni cuántos ni el desplazamiento, siendo la instrucción de la forma:

int info = pvm_pkstr(char *cadena)

Otra alternativa es usar la función:

int info=pvm_packf(const char *formato, ...)

que se invoca de la misma forma y con los mismos parámetros que el printf de C, por lo que pueden ignorarse tranquilamente las otras 13 funciones.

Bajo ningún concepto ningún mensaje puede ser mayor de 32 Mbytes.

C.6.9. Tipos de datos para empaquetado y desempaquetado

Tipo	Tipo C del 1er parámetro	1er Parámetro Fortran
byte	char *	BYTE1

Entero 1 byte	short *	BYTE1
Entero 2 byteS	int *	INTEGER2
Entero 4 bytes	long int *	INTEGER4
Flotante 4 bytes	float *	REAL4
Flotante 8 bytes	double *	REAL8
Complejo 8 bytes	float *	COMPLEX8
Complejo 16 bytes	double *	COMPLEX16
Cadena	Char *	STRING

C.6.10. Funciones para empaquetado y desempaquetado

Tipo	Empaquetado	Desempaquetado
Byte	pvm_pkbyte	pvm_upkbyte
Entero 1 byte	pvm_pkshort	pvm_upkshort
Entero 2 bytes	pvm_pkint	pvm_upkint
Entero 4 bytes	pvm_pklong	pvm_upklong
Flotante 4 bytes	pvm_pkfloat	pvm_upkfloat
Flotante 8 bytes	pvm_pkdouble	pvm_upkdouble
Complejo 8 bytes	pvm_pkcplx	pvm_upkcplx
Complejo 16 bytes	pvm_pkdcplx	pvm_upkdcplx
Cadena	pvm_pkstr	pvm_upkstr

C.6.11. Envío de datos

El envío se realiza con la función:

int info= pvm_send(int TID, int canal)

Donde manda a la tarea con el TID indicado por el canal indicado el contenido del buffer de emisión por defecto.

Para mandar el buffer por defecto al mismo canal de conjunto de tareas, puede emplearse la función:

int info = pvm_mcast(int *TIDs, int número, int canal)

donde el primer parámetro será una matriz de TIDs y el segundo el número de TIDs contenidos en la matriz.

Es importante tener en cuenta que las comunicaciones se realizan mandando el mensaje solamente al canal de una tarea o de un conjunto de tareas. Si una tarea está bloqueada escuchando otro canal, no escuchará el mensaje que se está enviando.

La emisión no es bloqueante para el emisor respecto a la tarea del receptor, pero si respecto al daemon del emisor. Esto significa que la tarea cliente quedará bloqueada en la emisión hasta que todo el mensaje sea transmitido vía TCP para el daemon. Después, la tarea continuará ejecutandose, y el servidor intentará contactar con el daemon de la tarea receptora.

C.6.12. Recepción de datos

Para recibir datos, se emplean dos pasos:

- Recibir el mensaje
- Desempaquetar el dato

Exactamente igual que el envío, se emplea un buffer, sólo que ahora no hay que limpiarlo antes de recibir cada dato.

Es importante recordar que el buffer de envío y de recepción al arrancar una tarea en PVM son distintos y tienen distintas instrucciones para ser activados.

C.6.13. Recepción del mensaje

Al recibir el mensaje, la tarea receptora queda bloqueada contactando con su servidor para un canal determinado. Según el servidor esté con un mensaje listo en dicho canal para dicha tarea o no, y según el modo de recepción, la tarea quedará bloqueada o no esperando un mensaje. De cualquier forma, mientras el mensaje es transmitido entre el daemon receptor y la tarea receptora, la tarea receptora es bloqueada. El mensaje es transmitido entre daemon receptor y tarea receptora vía TCP. Además de todo esto, la espera entre emisión del mensaje y recepción puede ser bloqueante o no según la función de recepción. Estas son:

int bufid=pvm_recv(int TID, int canal) : bloqueante. El receptor esperará a que la tarea TID le mande un mensaje por el canal indicado, entonces transferirá el mensaje al buffer de recepción por defecto (bufid).

int bufid=pvm_nrecv(int TID, int canal): no bloqueante. Si se encuentra en el servidor un mensaje de la tarea TID para el canal apropiado ya lista, se transfiere el mensaje de forma bloqueante al buffer de recepción por defecto (bufid). Si no, la función devuelve como bufid el valor 0, y no queda bloqueada.

int bufid=pvm_trecv(int TID,int canal, struct timeval *tiempo): bloqueante con temporalización. El receptor estará bloqueado durante el tiempo indicado, tiempo que se comporta como la recepción bloqueante. Transcurrido ese tiempo, pasa a comportarse como recepción no bloqueante, por lo que, si aún no llegó mensaje, devolverá el valor 0.

Una función interesante es pvm_probe, que verifica si ha llegado un mensaje determinado. Su comportamiento es exactamente igual que el de pvm_nrecv, sólo que no recibe el paquete; por lo que para recibirlo sigue siendo precisa una instrucción de recepción de paquete. Su sintaxis es:

int bufid = pvm_probe(int TID, int canal)

aún sin bajarlo, puede obtenerse información detallada del mensaje con la función pvm_bufinfo, que da información sobre el mensaje recibido antes de descargarlo; basta con combinar su uso con pvm_probe.

Cualquiera de las funciones de recepción antes mencionadas pueden tener como parámetro TID el valor -1, lo que significa cualquier tarea. Del mismo modo, un canal con valor -1 significa cualquier canal.

Envío y recepción entre dos tareas determinadas siempre se realizan en orden. Esto significa que si una tarea manda dos mensajes a otra, los mensajes serán recogidos por la tarea destinataria en el mismo orden en el que fueron mandados. Por otro lado, si dos tareas emiten cada una un mensaje a otra tercera, no hay forma de saber en qué orden serán recibidos por el receptor. Los mensajes recibidos de distintas tareas se pueden recibir fuera de orden.

C.6.14. Desempaquetado de datos

El desempaquetado de datos es exactamente igual que el empaquetado de datos, con sus catorce funciones y con todos sus parámetros iguales, solo que cambiando pvm_pk por pvm_upk. Es decir, el formato en once de ellas es:

int info = pvm_upkXXX(YYY *que, int cuantos, int desde_donde)

donde XXX es el tipo en PVM de lo que vamos a recibir (byte, cplx, long, etc.) e YYY el tipo en C relacionado, siendo la instrucción de la forma:

int info = pvm_upkstr(char *cadena)

Exactamente igual que con el empaquetado de datos, existe la función:

int info=pvm_upackf(const char *formato, ...)

que tiene el mismo formato que el scanf de C y será más cómoda, por lo que tampoco en este caso se necesita ninguna de las otras trece funciones.

Por último, es importante destacar que los datos deben ser desempaquetados en el mismo orden que fueron empaquetados.

C.6.15. Sincronización de barrera

Este mecanismo de sincronización tiene como idea básica no dejar pasar a ninguna tarea hasta que un número suficente de tareas esté esperando en la barrera. El punto de espera se define con la función pvm_barrier. El formato de la instrucción es:

int info=pvm_barrier(char *grupo, int cuantos)

Donde el primer parámetro es el grupo sobre el que se está realizando la función y el segundo cuántos tienen que esperar para pasar. En caso de que se emplee el grupo sólo para sincronía, puede indicarse en el segundo parámetro -1, lo que significa que tomará todos los miembros del grupo en ese momento (función pvm_gsize) y lo empleará como parámetro. Puede ocurrir que distintos procesos tengan distintos valores de pvm_gsize porque se ejecutan en distintos momentos, por lo que el -1 puede ser un desastre si no hay seguridad de que todos los miembros del grupo ya están registrados.

C.6.16. Salida de la PVM

La salida es bastante fácil, basta con ejecutar la función pvm_exit(). Desde el maestro podrían matarse los esclavos con pvm_kill, pero es un procedimiento bastante poco recomendable.

C.6.17. Variables de entorno modificables por el usuario

Existen un conjunto de variables de entorno que pueden modificarse. Estas son:

- PVM_ROOT: Ruta del directorio de instalación de la PVM.
- PVM_EXPORT: Nombre de las variables del sistema que serán heredadas a los hijos lanzados con spawn.
- PVM_DPATH: Ruta del directorio de instalación de la PVM en los esclavos, por defecto.
- PVM_DEBUGGER: Ruta del script de depuración empleado por spawn.

C.6.18. Variables de entorno no modificables por el usuario

Existen variables de entorno que no se pueden modificar, pero que es interesante conocer para poder leer sus valores. Estas son:

- PVM_ARCH: Nombre de la arquitectura de la máquina local.
- PVMSOCK: Dirección del socket del daemon de PVM local.
- PVMEPID: PID supuesto de la tarea que se va a lanzar con spawn esperado.
- PVMTMASK: Máscara de depuración de la libpvm.

C.6.19. Opciones del parámetro flag para lanzar tareas

- PvmTaskDefault: La PVM decide en qué máquina va a lanzar las tareas.
- PvmTaskHost: El parámetro host indica en qué máquina se van a lanzar las tareas.
- PvmTaskArch: El parámetro host indica en que arquitectura (conjunto de máquinas con la misma variable $PVM_ARCH) se van a lanzar las tareas.
- PvmTaskDebug: La tarea será lanzada junto con el depurador.
- PvmTaskTrace: Se mantendrá un histórico para las llamadas a las rutinas de la PVM para esa tarea.
- PvmMppFront: Las tareas serán lanzadas con un front-end de MPP.
- PvmHostCompl: Las tareas serán lanzadas en aquellos nodos que no cumplan las características indicadas.

Para indicar varias opciones, lo hacemos asociándolas con el operador ||.

C.6.20. Opciones de codificación del buffer

El buffer admite varias formas distintas de codificación, seleccionables empleando flags. Estos son:

- PvmDataRaw: Los mensajes serán mandados sin codificar en XDR. Si al desempaquetar no es entendido el formato, generará un error. No es buena idea si se cuenta con una arquitectura heterogénea.
- PvmDataDefault: Los mensajes serán mandados codificados con XDR.
- PvmDataInPlace: En el buffer tendremos punteros a datos y tamaños de datos, en lugar de datos. Acelera bastante algunas operaciones, pero hemos de tener cuidado con modificar los datos alegremente después de empaquetar.
- Para indicar varias opciones, se asocian con el operador ||. PvmDataRaw y PvmDataDefault al mismo tiempo no tienen mucho sentido.

C.6.21. Otras funciones de mantenimiento de tareas

PVM permite desde el código controlar las distintas tareas con un conjunto de funciones. Estas son:

- Mata la tarea con el TID indicado. Para salir es mejor pvm_exit que matarse a si mismo.

```
int rc = pvm_kill(int tid)
```

- Devuelve el TID del proceso que lanzó la tarea, o PvmNoParent si fue lanzada directamente por el usuario.

```
int tid = pvm_parent()
```

- Devuelve información sobre una tarea.

```
int status = pvm_stat(int tid)
```

C.6.22. Funciones para operar sobre el conjunto de máquinas

Así como se puede operar sobre el conjunto de máquinas desde un archivo de configuración, también puede hacerse desde el propio programa. Para operar sobre el conjunto de máquinas desde el programa las funciones son:

- Añade un conjunto de máquinas de la PVM.

```
pvm_addhosts(char **hosts, int nhost, int *infos)
```

- Elimina un conjunto de máquinas de la PVM.

```
pvm_delhosts(char **hosts, int nhost, int *infos)
```

- Devuelve PvmOk si la máquina indicada está en la PVM, PvmNoHost en cualquier otro caso. Útil para que la aplicación sea tolerante a fallos.

```
int pvm_mstat(char *host)
```

- Devuelve información acerca del estado de la máquina paralela virtual. nhost será el número de máquinas, narch el número de arquitecturas y la hosts será un puntero a una matriz con los datos de todos los nodos -TID, nombre, arquitectura y velocidad relativa-.

```
int pvm_config(int *nhost, int *narch, struct pvmhostinfo **hosts)
```

- Para la máquina paralela virtual.

```
int rc = pvm_halt()
```

- Arranca el daemon para la máquina donde se ejecuta la tarea.

```
int dtid = pvm_start_pvmd(int args, char **argv, int block)
```

- Devuelve el identificador de máquina de un proceso. Sólamente extrae el dato del TID sin comprobar ni que exista ni que esté activo.

```
int dtid = pvm_tidtohost(int tid)
```

C.6.23. Funciones para operaciones con buffers

Pueden realizarse operaciones con los buffers desde para emplear más de los buffers que hay por defecto. Para ello se emplean las funciones:

- Devuelve el identificador del buffer activo de recepción.

```
int bufid=pvm_getrbuf()
```

- Devuelve el identificador del buffer activo de emisión.

```
int bufid=pvm_getsbuf()
```

- Activa un buffer para emisión.

```
int bufid=pvm_setsbuf()
```

- Activa un buffer para recepción.

```
int bufid=pvm_setrbuf()
```

C.6.24. Otras operaciones con grupos de la PVM

Además de las operaciones con grupos básicas ya explicadas, disponemos de otras operaciones adicionales bastante interesantes. Estas son:

- pvm_gather: recoge los datos de todo un grupo para almacenarlos en una matriz. Lo opuesto a pvm_scatter.

- pvm_gettid: devuelve el TID de una tarea para un número de instancia.

- pvm_reduce: Aplica un operador de reducción -sumatoria, producto, máximo, mínimo y otros definidos por el usuario- a los miembros de un grupo, generando un solo escalar.

- pvm_scatter: distribuye datos de una tarea a las restantes de un grupo. Los datos están originariamente en una matriz, y se mandan los count primeros a la primera, los count siguientes a la siguiente y asi los restantes.

- pvm_getinst: devuelve el números de instancia de una tarea dentro de un grupo.

Las operaciones de grupo sólo se deben realizar con grupos estables. Por ello es recomendable bloquear con pvm_barrier después de la inscripción de las tareas en un grupo, para que sólamente pasen todas cuando todas estén inscriptas.