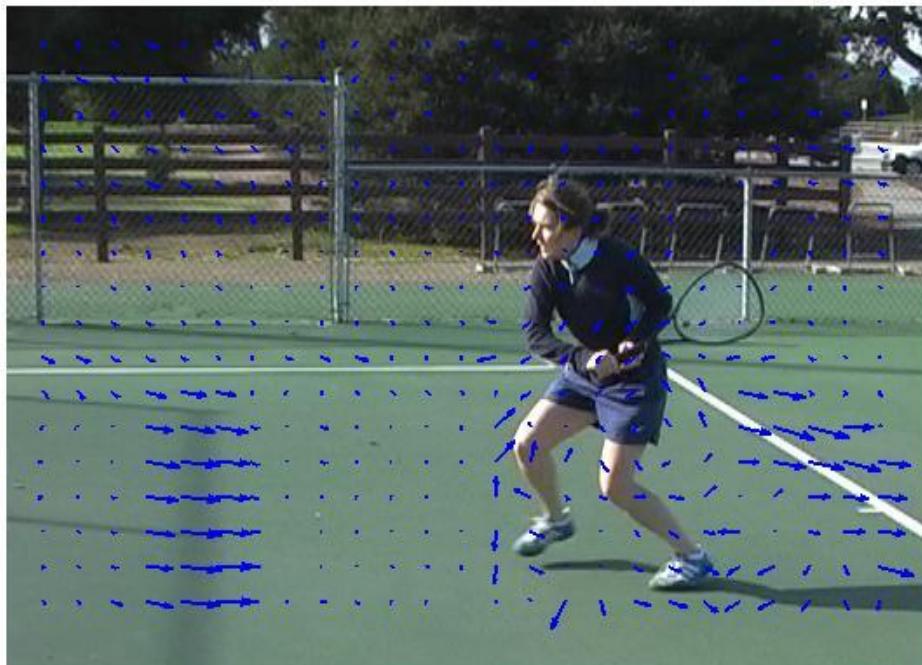


Práctica Flujo Óptico: LK + H&S

Método 1 : Lucas-Kanade - Método 2 : Horn y Schunck



Luis ROSARIO TREMOULET

01/04/2021

URJC - Máster Universitario en Visión Artificial

Sumario

Introducción	2
¿ Cómo se genera el flujo óptico ?	2
Ejemplos de flujos ópticos	3
Objetivo	3
Lucas-Kanade	4
Horn & Schunck	4
Codigo	5
Resultados : Lucas-Kanade	6
La importancia del tamaño de la imagen	6
El parametro “arrows_count”	9
Movimiento de cámara	10
Talla de la ventana	11
Resultados : Horn y Schunck	13
Velocidad de ejecución	13
Iteraciones	15
Lambda	16
Conclusión	18

Introducción

Este documento pretende resumir, explicar y comparar dos métodos de cálculo del flujo óptico que son : el método de Lucas-Kanade y el de Horn y Schunck.

El flujo óptico es el movimiento aparente de los objetos en una escena debido al movimiento relativo entre el observador y la escena. Suele utilizarse para seguir el movimiento de los objetos en una secuencia de vídeo y puede emplearse para estimar el movimiento de los objetos entre los fotogramas de un vídeo. Suele representarse como un campo vectorial bidimensional (en los resultados siguientes será representado por flechas azules).

¿ Cómo se genera el flujo óptico ?

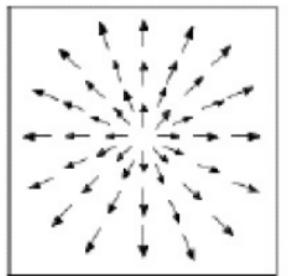
El propósito del flujo óptico es proyectar todos los movimientos 3D en un plano 2D.

El flujo óptico puede ser el resultado de :

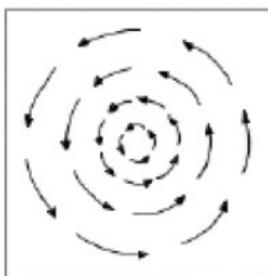
- un movimiento de objetos y/o de la cámara.
- un cambio en la iluminación o apariencia de los objetos (Por lo tanto, el movimiento percibido también depende del tipo de superficie de los objetos y la iluminación de la escena).
- un cambio en los parámetros intrínsecos de la cámara (por ejemplo, distancia focal, apertura, etc.).

Ejemplos de flujos ópticos

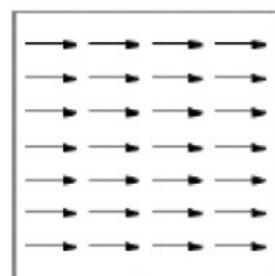
Ejemplos de diferentes flujos ópticos debido a los movimientos de la cámara :



zoom



rotación



translación

Objetivo

El objetivo de esta práctica es estimar, para cada píxel, un vector de

desplazamiento $\vec{V} = (v_x, v_y)$ expresando :

- La velocidad de desplazamiento de un píxel en la imagen
- La dirección en la que se mueve el píxel.

Solucionaremos este problema mediante 2 métodos :

- Método de Lucas-Kanade
- Método de Horn & Schunck

Lucas-Kanade

El método **Lucas-Kanade** (1984) es un método de intensidad determinista basado en ecuaciones de regresión lineal. Es un método local : no puede proporcionar el flujo dentro de una región uniforme.

Para que el método Lucas-Kanade funcione bien es necesario :

- Pequeños movimientos entre las dos imágenes dadas
- Iluminación constante
- Movimiento constante en una vecindad cercana (con un factor de suavización).

Horn & Schunck

El método de **Horn y Schunck** (1981) propone resolver el problema de apertura mediante una restricción de suavización. A diferencia de Lucas-Kanade, Horn y Schunck es una resolución global del problema de flujo óptico. Esto se debe a que la restricción de suavizado se aplica a toda la imagen.

Por lo tanto, hacemos las siguientes suposiciones :

- Pequeños movimientos entre las dos imágenes dadas
- Iluminación constante
- Los desplazamientos en una vecindad son similares (restricción de suavidad).

Al contrario de Lucas-Kanade que es un método determinista, basado en ecuaciones de regresión lineal. El método Horn & Schunk es un método iterativo, basado en un término de regularización.

Código

Para resolver el problema de flujo óptico, he utilizado el software MATLAB.

Dentro de mi proyecto hay :

- Una carpeta con las imágenes/vídeos utilizados
- *main_lucaskanade.m* (algoritmo Lucas-kanade)
- *main_horn_schunk.m* (algoritmo Horn & Schunck)
- *get_derivatives.m* (calcula las derivadas)
- *choose_image.m* (elige las imágenes sobre las que trabajar)

Cada método tiene sus propios parámetros configurables que pueden dividirse en dos categorías: los que son importantes para el algoritmo (por ejemplo : número de iteraciones, lambda, etc.) y los que no se utilizan en el algoritmo (por ejemplo, las fotos elegidas, el tamaño y el color de las flechas que representan el flujo óptico, etc.). En función de estos parámetros, los resultados serán más o menos cualitativos y rápidos.

Resultados : Lucas-Kanade

Los resultados dependen en gran medida de los parámetros escogidos y la calidad de las fotos. En primer lugar, necesitamos dos fotos que no estén muy separadas en el tiempo. Luego para los parámetros es cuestión de ajustar el tamaño de la ventana y de un parámetro que llamaré “arrows_count”.

La ventana de integración depende de la "velocidad" de la imagen, si el espacio temporal es muy grande entre las dos imágenes, entonces habrá que escoger un tamaño de ventana más grande de lo habitual y viceversa.

El segundo parámetro no cambia nada al resultado, si no que sirve para la representación del flujo óptico en la imagen de salida. Mas es grande, menos “flechas” habrán pero representaran los flujos más importantes (y serán más visibles). En concreto reduce el tamaño de la U y V final, utilizados en la función quiver().

La importancia del tamaño de la imagen

Ejemplo 1 : Mono saltando





Indicaciones :

- Algoritmo = Lucas-Kanade
- **Parámetro resize_value = 1**
- Parámetro arrows_counts = 20
- Parámetro ventana_original = 45
- Movimiento de cámara = Si
- Talla imagen = 720x432 pixels
- **Tiempo = 50.776079 segundos**

Aquí podemos ver los resultados de una prueba clásica. Podemos ver que el algoritmo es bastante bueno cuando se le pide que genere resultados precisos, sin embargo tarda mucho tiempo. Para ello podemos reducir el tamaño de la imagen por 2 para acelerar el proceso.

Ejemplo 2 : Mono saltando imagen 2 veces más pequeña



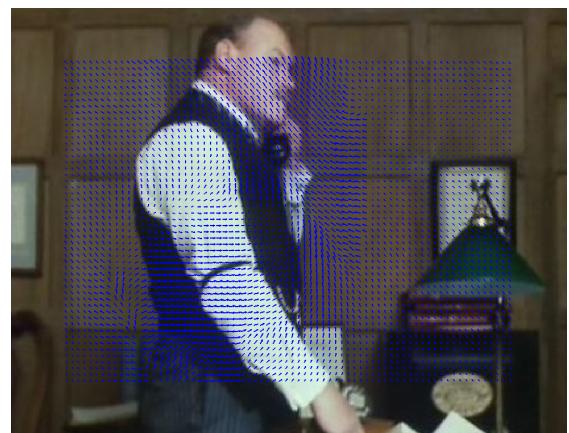
Indicaciones :

- Algoritmo = Lucas-Kanade
- **Parámetro resize_value = 0.5**
- Parámetro arrows_counts = 20
- Parámetro ventana_original = 45
- Movimiento de cámara = Si
- Talla imagen = 720x432 pixels
- **Tiempo = 5.934386 segundos**

Aquí hemos reducido el parámetro "resize_value" que divide el tamaño de la imagen y de la ventana por dos. Al reducir el parámetro "resize_value" podemos ver que el algoritmo es mucho más rápido. Pero al reducir el tamaño de la imagen, empieza a tener mucho ruido, lo que lleva a un pequeño “ruido” de flujo (circulo rojo).

El parametro “arrows_count”

Ejemplo 3 : Escena película



Indicaciones :

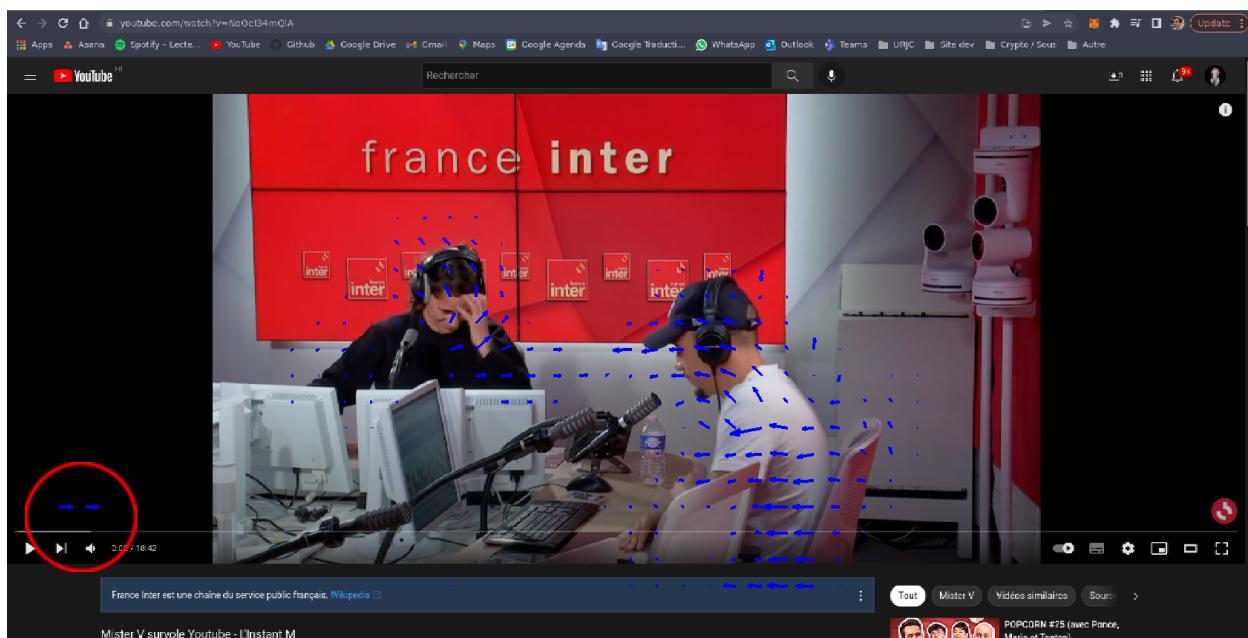
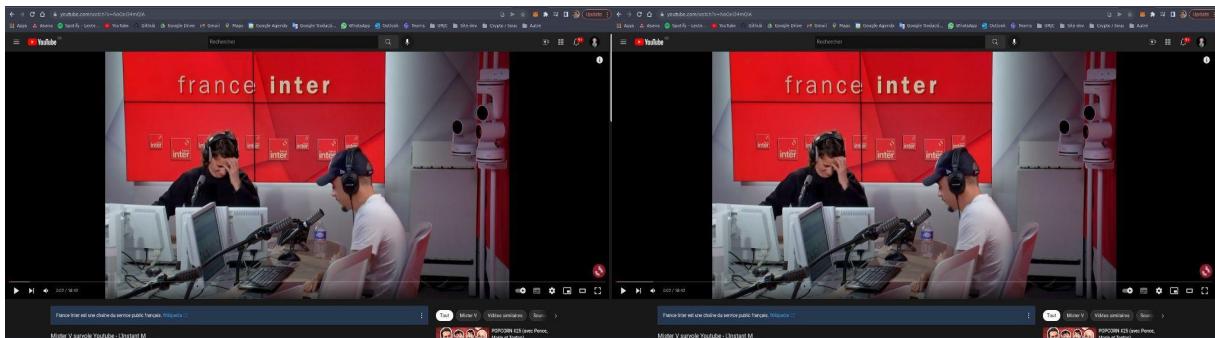
- Algoritmo = Lucas-Kanade
- Parámetro resize_value = 0.5
- **Parámetro arrows_counts = 40**
- Parámetro ventana_original = 45
- Movimiento de cámara = Si
- Talla imagen = 450x350 pixels
- Tiempo = 2.327648 segundos

Algoritmo = Lucas-Kanade
Parámetro resize_value = 0.5
Parámetro arrows_counts = 5
Parámetro ventana_original = 45
Movimiento de cámara = Si
Talla imagen = 450x350 pixels
Tiempo = 2.535492 segundos

Aquí podemos ver la influencia del parámetro "arrows_count". Como se ha dicho antes, no tiene ningún efecto sobre el algoritmo, sino sólo sobre el resultado visualizado. Podemos ver aquí gracias a la imagen de la derecha (con un débil "arrows_count") que hay un movimiento de cámara además del movimiento del actor, lo que no se podría determinar exactamente con la imagen de la izquierda.

Movimiento de cámara

Ejemplo 4 : Radio en directo



Indicaciones :

- Algoritmo = Lucas-Kanade
- Parámetro resize_value = 0.5
- Parámetro arrows_counts = 40
- Parámetro ventana_original = 45
- **Movimiento de cámara = No**
- Talla imagen = 1913x970 pixels
- Tiempo = 35.135464 segundos

Por primera vez tenemos un ejemplo en el que no hay movimiento de cámara. Podemos confirmarlo porque sólo hay flujo óptico en los presentadores de la radio y en la barra de carga de youtube. Podemos concluir que el algoritmo funciona bastante bien.

También podemos ver que al aumentar el tamaño de la imagen se incrementa drásticamente el tiempo de ejecución.

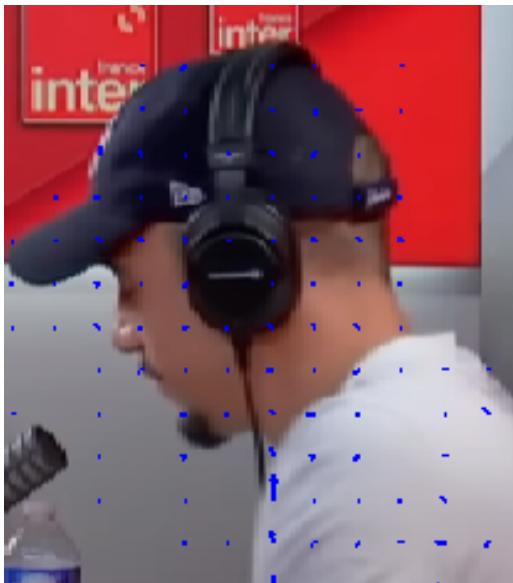
Talla de la ventana

Ejemplo 5 : Radio en directo (zoom)



Indicaciones :

- Algoritmo = Lucas-Kanade
- Parámetro resize_value = 0.5
- Parámetro arrows_counts = 20
- **Parámetro ventana_original = 45**
- Movimiento de cámara = No
- Talla imagen = 1913x970 pixels
- Tiempo = 31.311035 segundos



Indicaciones :

- Algoritmo = Lucas-Kanade
- Parámetro resize_value = 0.5
- Parámetro arrows_counts = 20
- **Parámetro ventana_original = 5**
- Movimiento de cámara = No
- Talla imagen = 1913x970 pixels
- Tiempo = 9.862576 segundos

Aquí podemos ver la importancia del tamaño de la ventana. Este parámetro nos permite saber con cuántos píxeles vecinos debe compararse el píxel calculado. Este parámetro varía dos cosas: el tiempo (cuanto más pequeña sea la ventana, más corto será el tiempo de ejecución) y el resultado. Si la ventana es demasiada pequeña en comparación con el desplazamiento de la imagen, no se detectará ningún movimiento o muy poco (como se puede ver en la segunda imagen del ejemplo 5). Por el contrario, si la ventana es demasiado grande, los resultados serán erróneos.

Resultados : Horn y Schunck

Como el algoritmo de Lucas-Kanade, éste también requiere imágenes con poco espacio temporal. También tiene algunos parámetros muy importantes que influyen en el resultado. En primer lugar, al ser un método de iteración, hay que elegir el número de iteraciones del algoritmo (entre 10 y 100). Además tenemos un parámetro lambda que influirá en el valor de U y V (entre 0,1 y 60). Para terminar con los parámetros importantes también podemos elegir las inicializaciones de U0 y V0, Si se dispone de ellos, el flujo convergerá más rápido y, por tanto, necesitará menos iteraciones, por defecto son ceros.

Además de los parámetros importantes, también están los necesarios para mi propia forma de desarrollar este método. Al igual que L-K tenemos el parámetro "arrows_count" y muchos otros que influyen en la visualización del resultado

Velocidad de ejecución

Ejemplo 6 : Tenis





Indicaciones :

- Algoritmo = Horn y Schunck
- Parámetro lambda = 4
- Parámetro iteraciones = 100
- Parametro “arrows_count” = 15
- Movimiento de cámara = Si
- Talla imagen = 530x380 pixels
- **Tiempo = 0.380844 segundos**

Podemos ver que el algoritmo L-K es mucho más lento que el algoritmo H&S. De echo, para una imagen de tamaño 530x380 píxeles y con un número de iteraciones de 100, H&S es 10 a 30 veces más rápido que L-K (según mis cálculos y ejemplo 2 y 3).

Iteraciones

Ejemplo 7 : Tenis con pocas iteraciones



Indicaciones :

- Algoritmo = Horn y Schunck
- Parámetro lambda = 4
- **Parámetro iteraciones = 10**
- Parametro “arrows_count” = 15
- Movimiento de cámara = Si
- Talla imagen = 530x380 pixels
- Tiempo = 0.115124 segundos

Al reducir el número de iteraciones, podemos concluir dos cosas. En primer

lugar, podemos ver que el tiempo de ejecución es más rápido (lo cual es bastante lógico). Y en segundo lugar podemos deducir comparando los dos resultados que el que tiene menos iteraciones, tiene menos precisión(hay menos detalles). Por lo tanto, podemos concluir que cuantas más iteraciones haya, más preciso será el resultado (hay que tener cuidado de no generar flujos “inexistentes” debido a demasiadas iteraciones).

Lambda

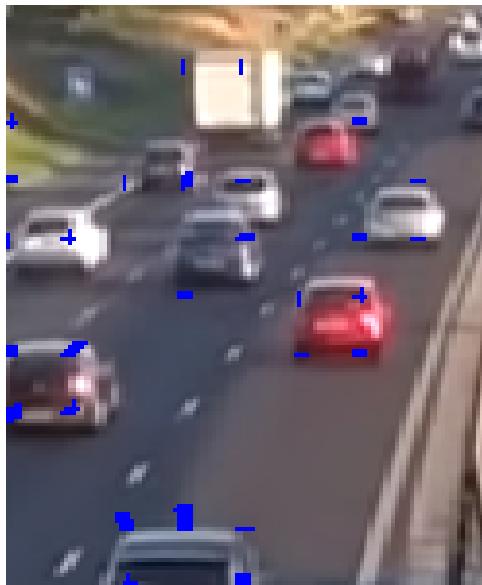
El lambda es un parámetro del algoritmo Horn & Schunk. Influirá en el resultado de la U y la V en el cálculo iterativo, como podemos ver en las fórmulas siguientes. Generalmente tiene un valor entre 0.2 y 60.

$$u^k = u^{-k-1} - I_x \frac{I_x u^{-k-1} + I_y v^{-k-1} + I_t}{\lambda^2 + I_x^2 + I_y^2}$$

$$v^k = v^{-k-1} - I_y \frac{I_x u^{-k-1} + I_y v^{-k-1} + I_t}{\lambda^2 + I_x^2 + I_y^2}$$

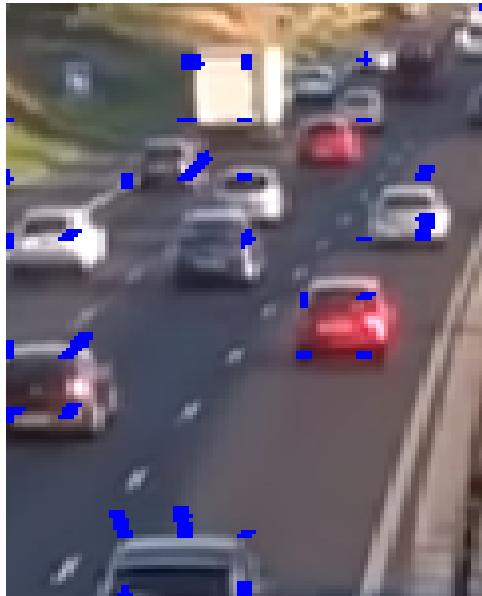
Ejemplo 8 : Tráfico de coches





Indicaciones :

- Algoritmo = Horn y Schunck
- Parámetro lambda = 0.1
- Parámetro iteraciones = 10
- Parametro “arrows_count” = 15
- Movimiento de cámara = No
- Talla imagen = 1339x693 pixels
- Tiempo = 0.309979 segundos



Indicaciones :

- Algoritmo = Horn y Schunck
- Parámetro lambda = 60
- Parámetro iteraciones = 10
- Parametro “arrows_count” = 15
- Movimiento de cámara = No
- Talla imagen = 1339x693 pixels
- Tiempo = 0.337544 segundos

Podemos ver que el lambda es un parámetro importante en la resolución del flujo óptico. Aquí podemos ver que cuanto más grande es la lambda, más sensible es a los pequeños movimientos (el resultado mostrado es del fondo de la imagen, donde hay poco movimiento). También podemos ver que, independientemente del tamaño de la imagen, el tiempo de resolución sigue siendo muy rápido en comparación con L-K.

Conclusión

Los métodos Lucas-Kanade y Horn&Schunk son métodos que ayudan a obtener el flujo óptico entre dos imágenes (también pueden utilizarse para vídeos, ya que son secuencias de imágenes). Tras varias pruebas podemos deducir que ambos métodos funcionan muy bien cuando se respetan estos 5 criterios :

- Poco espacio temporal entre las dos imágenes (poco movimiento)
- Cuando la iluminación no cambia (o muy poco)
- Cuando no hay oclusión
- Cuando los objetos son rígidos
- Cuando hay poco ruido

En la carpeta "Dataset" hay imágenes/vídeos adicionales para probar todas estas limitaciones.

La primera diferencia entre estos dos métodos es la forma de resolver el problema. Uno lo resuelve de forma iterativa (basado en un término de regularización) : Horn & Schunk, el otro lo resuelve de forma determinista, basado en ecuaciones de regresión lineal. Por lo tanto, para el método H & S habrá que determinar un número de iteraciones.

Otra diferencia es el número de parámetros que hay que ajustar para que el algoritmo funcione. De hecho, en mis programas podemos contar 13 parámetros para H & S frente a 8 para L-K (también hay parámetros no útiles para la solución del problema).

A pesar de un mayor número de parámetros en H&S, este método sigue siendo extremadamente más rápido que L-K.

Sin embargo, estos dos algoritmos dan muy buenos resultados si se respetan ciertos criterios.