# Visión por Computadora I

Ing. Maxim Dorogov

(mdorogov@fi.uba.ar)

Laboratorio de Sistemas Embebidos -FIUBA

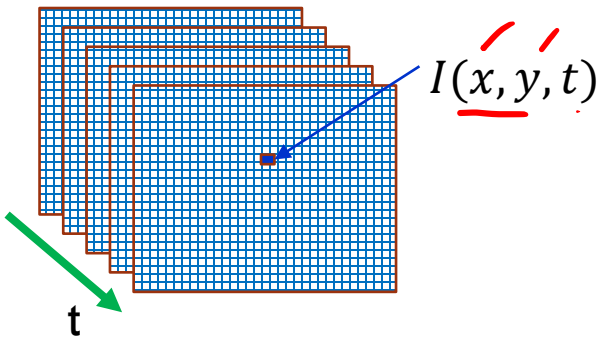
FACULTAD DE INGENIERIA
Universidad de Buenos Aires

# VIDEO DIGITAL

## Video

Secuencia de imágenes capturadas a lo largo del tiempo

- Nuestra señal suma una nueva variable, el tiempo.
- **Frame rate**: Cantidad de fotogramas por segundo (FPS)
- Generalmente los frames de la fuente están generados a tiempos regulares y constantes (60Hz, 30Hz, 24Hz, etc.)
- Una vez dentro del pipeline se busca procesar los frames, a tiempo constante y lo mas rápido posible, para mantener el frame rate de la fuente a la salida (proc. en tiempo real)




**Algunos frameworks:**

- **OpenCV**: Captura, procesamiento y transmisión de streams de datos en tiempo real.
- **GStreamer**: Adquiere, procesa y transmite multimedia mediante elementos agrupados en pipelines, esta escrito en C. Intel, Nvidia, AWS Kinesis, etc… mantienen plugins para inferencia, procesamiento y streaming.
- **FFMPEG**: A diferencia de gstreamer, al agregar una modificación se debe recompilar todo desde cero. Es menos flexible al momento de agregar elementos custom.


Esquema elemental de un pipeline de gstreamer o ffmpeg

gstreamer
open source multimedia framework

FFMPEG

# VIDEO DIGITAL

## Conceptos

**Fuente:** Archivo físico (Ej: pepito.mp4), red via streaming:

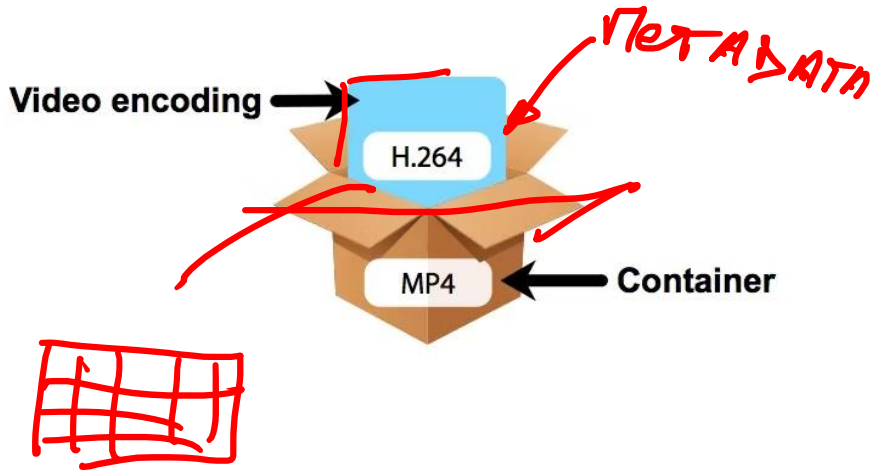
- ❑ RTSP (Real Time Streaming Protocol)
- ❑ RTMP (Real Time Messaging Protocol - Adobe)
- ❑ HTTP/HLS (Apple 2009).
- ❑ Etc..

| Codec | Container |
|---|---|
| H.264 or H.265 | MP4 or MOV |
| ProRes | MOV |
| DNxHD or DNxHR | MXF or MOV |
| H.264 or H.265 | 3GP, 3G2 or MP4 |
| H.264 or H.265 | MOV |

CODECs mas comunes y su compatibilidad con algunos containers.

**CODEC:** Codificador/Decodificador. Es el algoritmo que se utiliza para decodificar el video y acceder a los datos (matriz RGB, audio, etc…). Algunos ejemplos: H264, H265,H262/MPEG2, M-JPEG, DNxHD, y muchos otros…

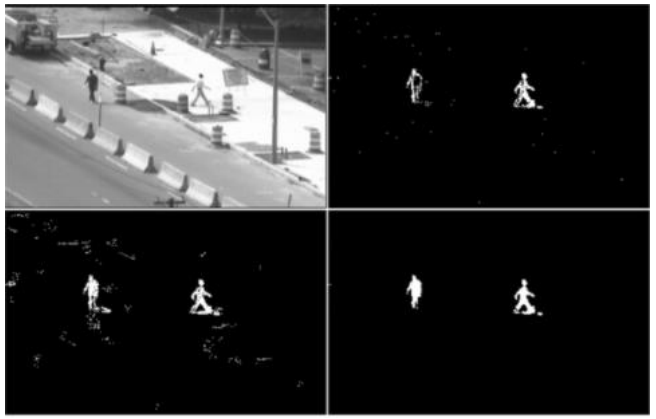
**Container:** Es el elemento que almacena el video codificado junto con la metadata. Ejemplos: AVI, MOV, MP4,3GP,etc…




**Archivo de video:** CODEC + Container

- En aplicaciones de Computer Vision es común utilizar H264 o H264+ con el container MP4 debido a su eficiente compresión en fuentes de alta resolución permitiendo streamings a +60 FPS.
- Tanto gstreamer como ffmpeg permiten transformaciones entre diversas fuentes, codecs y containers.

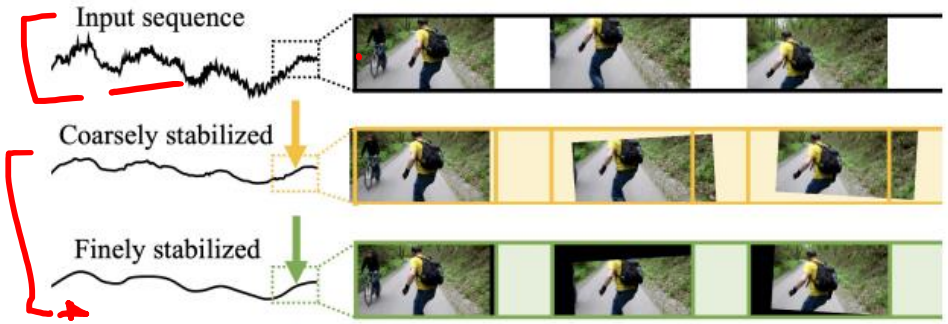
# PROCESAMIENTO DE VIDEO

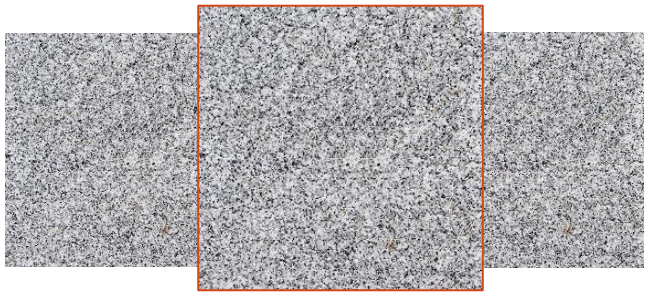


## Aplicaciones:

1. Separación de fondo (background substraction)
   - Separar el fondo estático del los objetos en movimiento
2. Segmentación por movimiento en espacio/tiempo
   - Segmentar el video en objetos "coherentemente" en movimiento
3. Optical Flow
   1. Estabilización de imagen
   2. Detección de movimiento
   3. Edición: Detección de tomas, cortes por escena inteligentes.
4. Tracking
5. Corrección de anomalías (ej: producidas por rolling-shutter)
6. Aprendizaje de modelos dinámicos:
   - Detección y clasificación de accidentes (choques de vehículos, caídas, etc…)

En ocasiones es mas simple realizar comparaciones visuales sobre video que en imágenes estáticas…

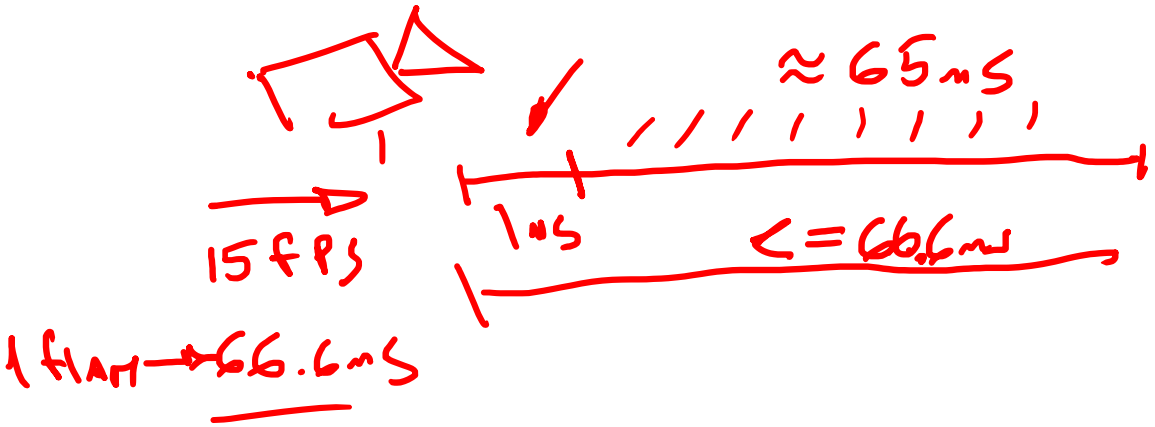


# VIDEO DIGITAL

## Ejercicio

1. **En un sistema de visión con procesamiento en tiempo real se emplea una cámara que posee un frame rate de 15fps. Calcular cual es el tiempo máximo que se puede destinar al procesamiento de cada frame sabiendo que la operación de lectura de un frame desde la cámara demora 1ms.**



# BACKPROJECTION (RETROPROYECCIÓN)

## Retroproyección de histogramas

- Permite trabajar en un espacio de características apropiado para tracking con meanshift y camshift
- En términos estadísticos nos dice que tan probable es que un pixel se corresponda con la clase que estoy buscando.

## Algoritmo

- Se toma una ROI que contenga el objeto de interés, en un espacio de color determinado, y se calcula el histograma 1D o 2D.
- En cada nuevo frame calculo el histograma (respetando canales, espacio de color y la cantidad de bins del histograma patrón)
- Para cada pixel p(i,j) de la nueva imagen/roi obtengo su bin en el histograma y voy a buscar el valor que le correspondería a ese bin en el histograma patrón.
- Ese valor se guarda en una nueva imagen (imagen de retroproyección) en la posición (i, j)
- La imagen de retroproyección pasa a ser el nuevo espacio de características.

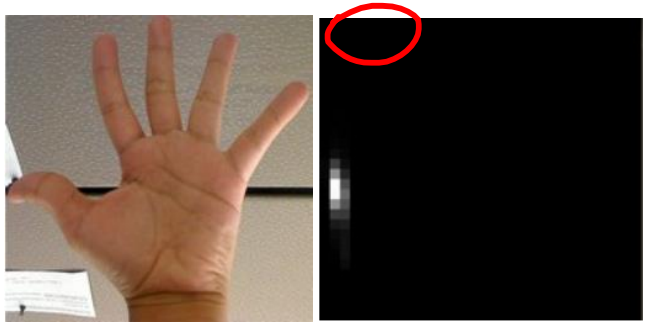
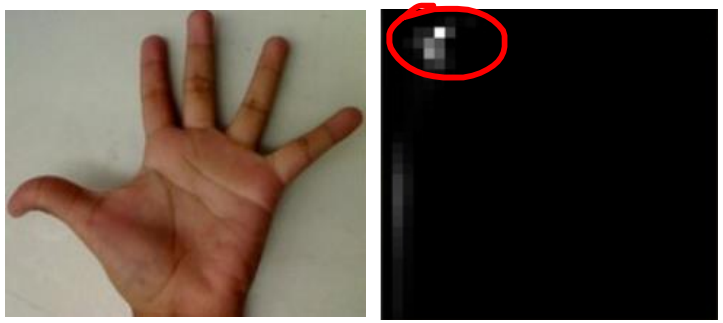
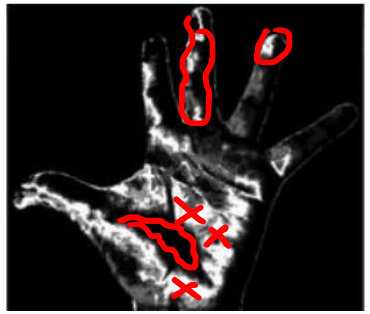


Imagen e histograma patrón



Nueva imagen y su histograma



Retroproyección

# MEANSHIFT (DETECCIÓN Y TRACKING)

## Motivación

- Necesidad de detectar y seguir objetos sobre una secuencia de imágenes en tiempo real.
- Es difícil de lograr a partir de un único template en objetos de mucha variabilidad (non-rigid objects), necesitamos actualizarlo a medida que transcurre la escena.

## Algoritmo

- Se toma una ROI que contenga el objeto a detectar
- Extracción de características: Se convierte la ROI a HSV se calcula el histograma para el Hue.
- Se calcula la retroproyección del histograma sobre el frame actual
- Meanshift busca la zona donde se maximiza la retroproyección de manera iterativa
- Se actualiza la ROI y se repite el proceso sobre un nuevo frame

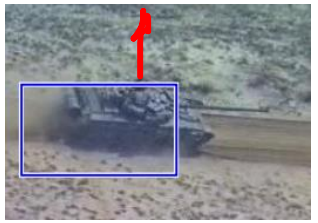
# MEANSHIFT (DETECCIÓN Y TRACKING)

## Pros

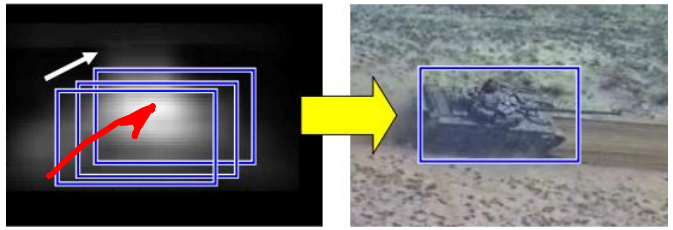
- Tracking en tiempo real
- Bajo uso de recursos, fácil de implementar
- Se puede combinar con otras técnicas (meanshift-Kalman) para robustecer el algoritmo
- Invariante a rotación (siempre que no cambie la distribución de características)

## Contras

- Dependiente de la cantidad de bins del histograma
- Se asume que los desplazamientos son en un entorno de la ventana
- Falla cuando hay oclusión total o parcial del objeto a detectar
- No es invariante a la escala
- La performance se ve afectada en escenarios con mucho ruido



Frame actual y ultima posición detectada



Retroproyección y corrección de la posición del objeto corregida

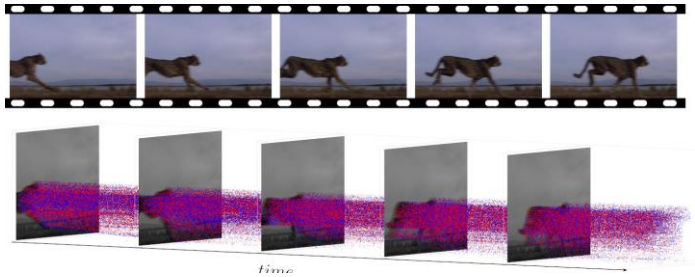
# DETECCIÓN DE MOVIMIENTO

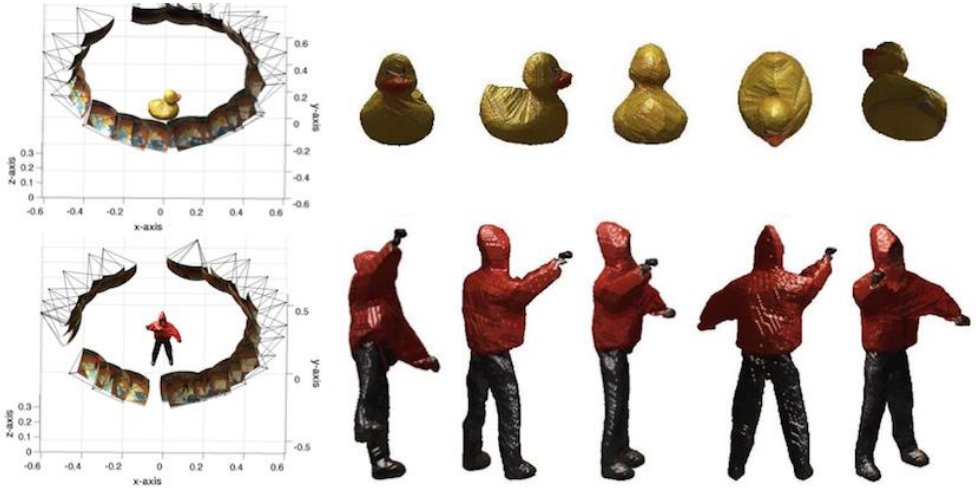
ASIC - FPGA

## Hardware Especifico

Cámaras por eventos (event/neuromorphic cameras)

Únicamente transmiten los pixeles que tuvieron cambios en su nivel de intensidad.



(a) Camera positions of the selected video frames for Duck and Action Man.



(b) Rendered views of the estimated 3D geometry.

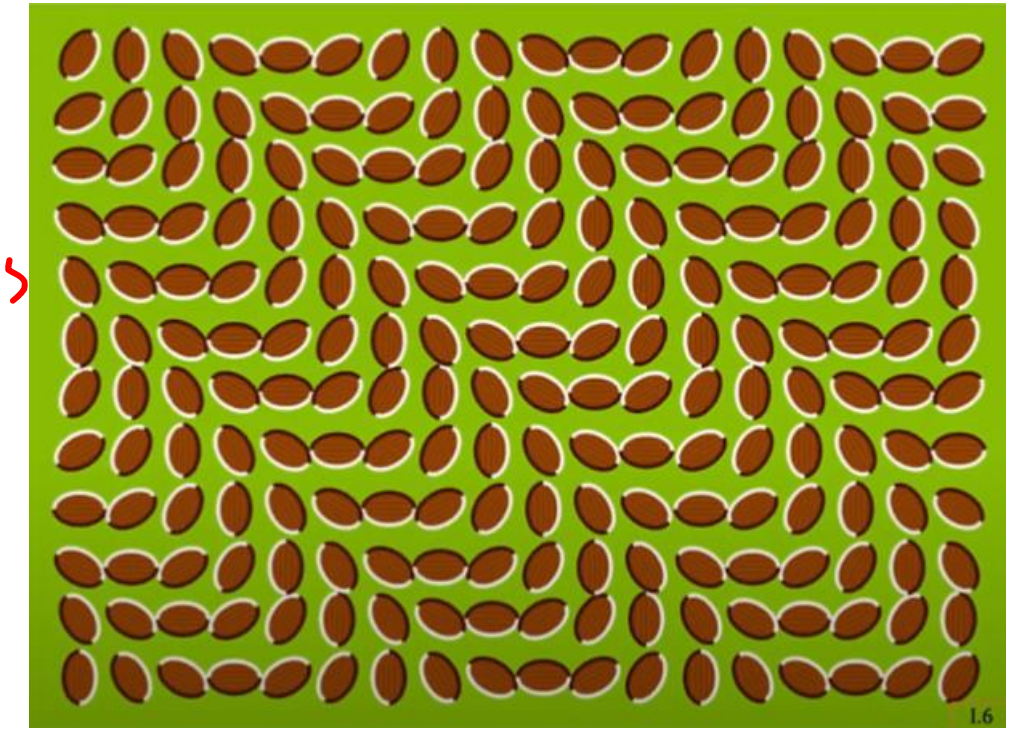
Movimiento de cámara alrededor de un objeto para generación de puntos 3D

## Estimación de movimiento: Optical Flow.

1. **Métodos basados en características** ✓
   - Extraer características visuales (esquinas, keypoints, etc.) y seguirlas a través de los cuadros.
   - Esto da lugar a los campos de movimiento escasos (basados solo en las características que son buenas para el seguimiento). Aún así es un seguimiento robusto.
   - Adecuados cuando los movimientos son grandes (decenas de píxeles)
2. **Métodos directos o "densos" (Dense Flow)** ✓
   - Buscan recuperar el movimiento de cada píxel a partir de variaciones espacio-temporales de los niveles de brillo
   - Da lugar a campos densos, pero sensibles a variaciones de apariencia
   - Adecuados cuando los movimientos son pequeños.

# DENSE FLOW

- **Flujo óptico**
  - Movimiento aparente de objetos o superficies
  - Si no hay cambios de intensidad relativos no se puede predecir (parte blanca)
- ¿Cómo estimamos el movimiento de $I(x, y, t)$ a $I(x, y, t + 1)$?
  - Buscamos resolver el problema de esta correspondencia.
  - Dado un pixel en $I(x, y, t)$ buscar por píxeles cercanos del mismo "color"
  - Este es el problema de "flujo óptico"



**Asumimos:**

- Constancia de color: Un punto en $I(x, y, t)$ se parece a un punto en $I(x', y', t + 1)$
  - Para imágenes en tonos de gris hablamos de "constancia de brillo"
- Pequeños movimientos: Se asume que los puntos no se mueven muy lejos entre $t$ y $t + 1$

# RESTRICCIONES DEL FLUJO ÓPTICO

- Restricción de constancia de brillo

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) \quad (1)$$

- Restricción de movimiento (pequeños desplazamientos)
  - 1 píxel o menos (las cosas cambian "suavemente")
  - Podemos escribir una aproximación de Taylor

$$I(x + \Delta x, y + \Delta v, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + \cdots$$

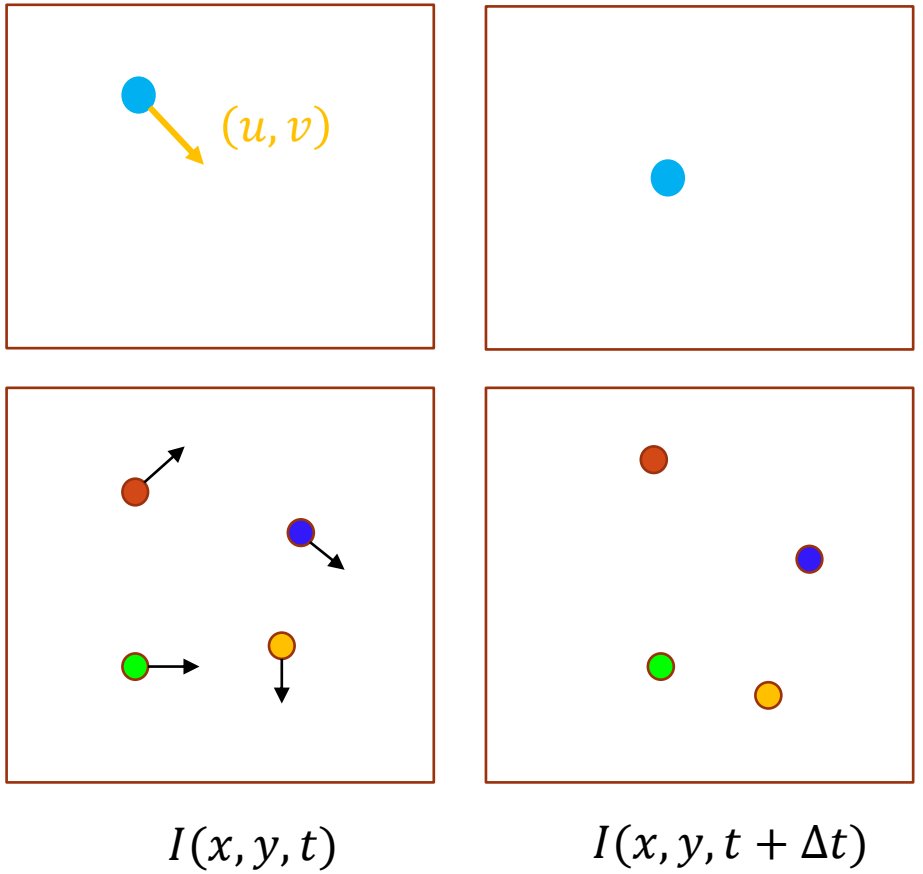
$$I(x + \Delta x, y + \Delta y, t + \Delta t) \approx I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t \quad (2)$$

- Restamos ambas ecuaciones y dividimos por $\Delta t$

$$\boldsymbol{I_t + I_x u + I_y v = 0}$$

Donde:

$$I_x = \frac{\partial I}{\partial x}; I_y = \frac{\partial I}{\partial y}; I_t = \frac{\partial I}{\partial t}; u = \frac{\Delta x}{\Delta t}; v = \frac{\Delta y}{\Delta t}$$

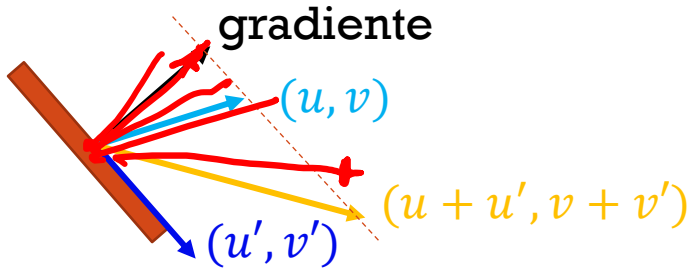


# RESTRICCIONES DEL FLUJO ÓPTICO

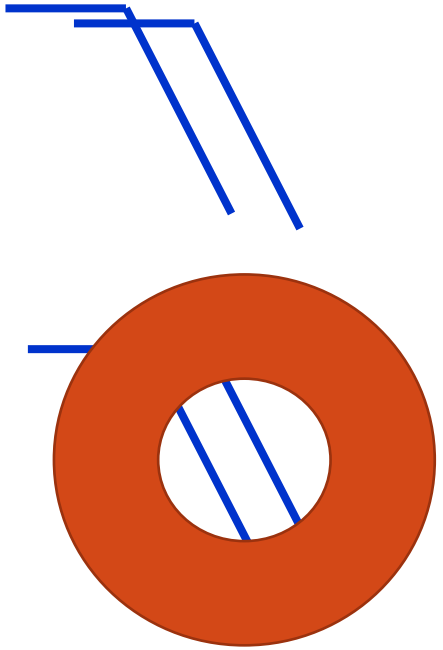
- Esta ultima se conoce como ecuación de restricción de brillo constante

$$I_t + I_x u + I_y v = 0$$

- ¿Cuántas incógnitas y ecuaciones tenemos por píxel? → **2 incógnitas** $(u, v)$…pero **una sola ecuación**!
- La componente de movimiento perpendicular al gradiente (paralelo al borde) no puede determinarse. Es decir, si $(u, v)$ satisface la ecuación, también lo hace $(u + u', v + v')$.



# PROBLEMA DE APERTURA



- **Enfoque global: Smooth Optical Flow (Horn y Schunk)**
  - Plantean el error en el flujo óptico

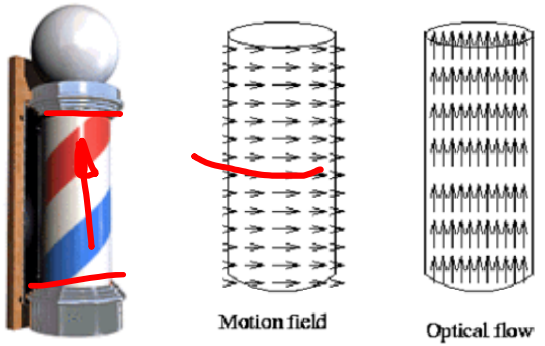
$$E = \iint \left[ (I_x u + I_y v + I_t)^2 + \alpha^2 (\|\nabla u\|^2 + \|\nabla v\|^2) \right] \mathrm{d}x\mathrm{d}y$$

Es decir, se asume que el campo de velocidad tiende a variar lentamente a lo largo de la imagen. Se penalizan cambios bruscos de $u$ y $v$

Se buscan los (u,v) en cada punto (píxel) que minimicen:

$$e = e_s + \lambda e_c$$

El parámetro $\lambda$ permite darle mayor o menor peso a la restricción de consistencia de brillo que a la de transiciones suaves y viceversa. Su definición será en base a cuánto le confiemos a los datos en cada caso.



# LUCAS-KANADE

- **Enfoque local:** La idea es imponer más restricciones *locales* a cada píxel
- Se asume que el flujo de velocidades es suave localmente (en píxeles vecinos)
- Tanto que se asume que píxeles vecinos (por ejemplo en una ventana de 5x5) tienen el mismo (u,v)

$$0 = I_t(p_i) + \nabla I(p_i).[u, v]$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

$$\underset{(25 \times 2)}{A} \quad \underset{(2 \times 1)}{d} = \underset{(25 \times 1)}{b}$$

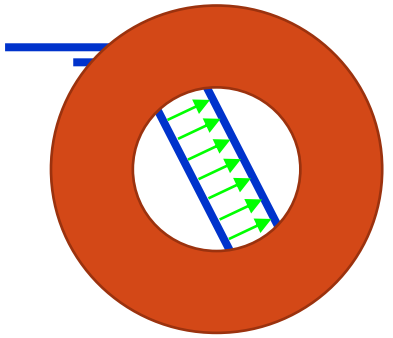
Es decir, pasamos a tener 25 ecuaciones por píxel!

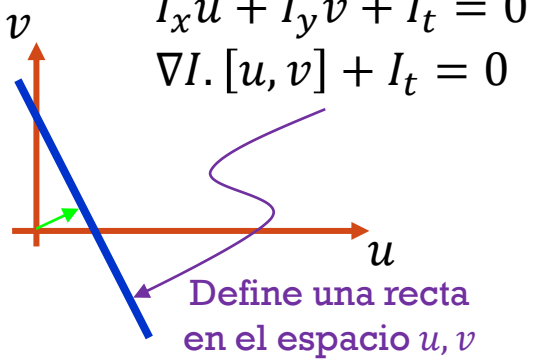
- ¿Cómo lo resolvemos? → por mínimos cuadrados, minimizando $\|Ad - b\|^2$

$$(A^t A)d = A^t b$$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

Estas sumatorias son sobre todos los píxeles en la ventana de $K \times K$



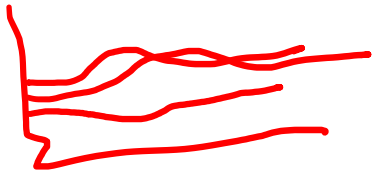


# LUCAS-KANADE

- ¿Cuándo es este sistema resoluble?
  - Cuando $A^tA$ es inversible
- Entonces $A^tA$ debe estar bien condicionada. Esto puede verse a través de la relación entre sus autovalores
  - $\lambda_1/\lambda_2$ no debe ser muy grande (considerando a $\lambda_1$ como el autovalor más grande)
- ¿A qué nos recuerda esto?

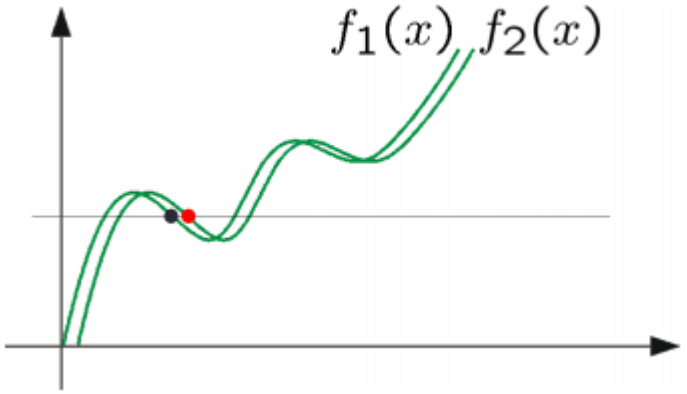
$$A^tA = \begin{bmatrix} \sum I_xI_x & \sum I_xI_y \\ \sum I_xI_y & \sum I_yI_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \quad I_y] = \sum \nabla I (\nabla I)^t$$

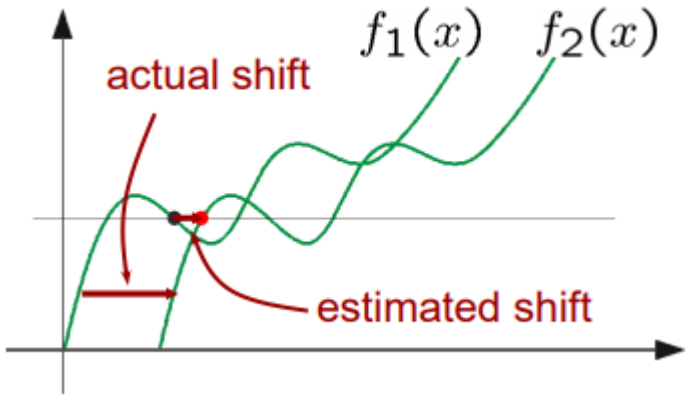
- Mismo criterio que el detector de esquinas de Harris. $M = A^tA$ es la matriz de momentos de orden 2.
  - Los autovectores y autovalores de M se relacionan con la dirección y magnitud del borde.
  - Recordando, algo era una buena esquina cuando $\lambda_1$ y $\lambda_2$ eran razonablemente grandes y comparables.
- ¿Qué pasa en imágenes color RGB?
  - En la misma ventana de $5 \times 5$ tendríamos $25 \times 3 = 75$ ecuaciones por píxel!
  - ¿Y si tuviésemos una "ventana de un solo píxel"?¿No podríamos resolver el sistema de dos incógnitas $(u, v)$ pero ahora tres ecuaciones?

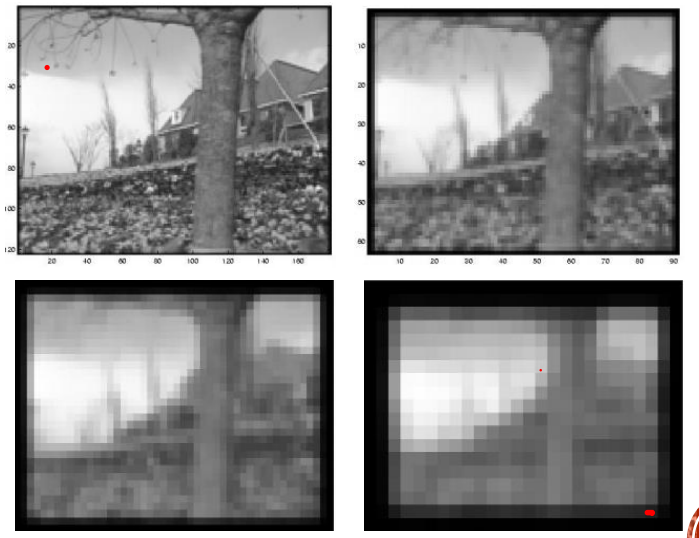


# LUCAS-KANADE

- En la práctica es lógico tener desplazamientos grandes de píxeles (pérdida de variaciones locales)
- Ademas tenemos una ambigüedad debida al aliasing temporal de la imagen donde muchos píxeles pueden tener el mismo nivel de intensidad
- Para superar esta situación podemos hacer una estimación de grueso a fino. Reduciendo la resolución!



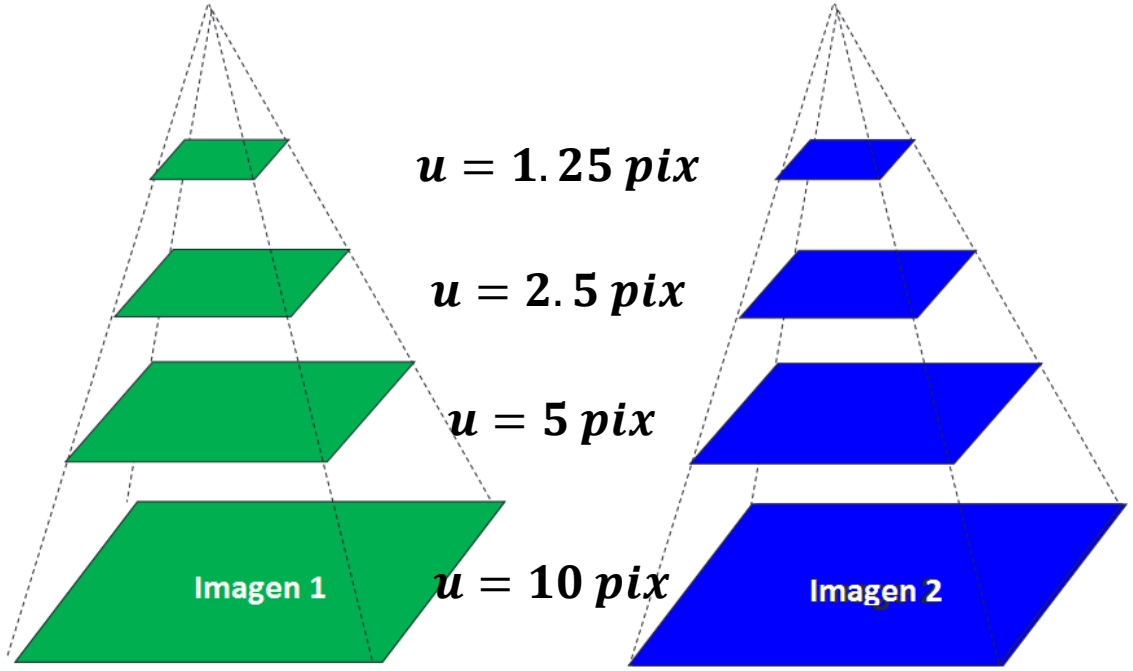
# LUCAS-KANADE POR JERARQUIAS

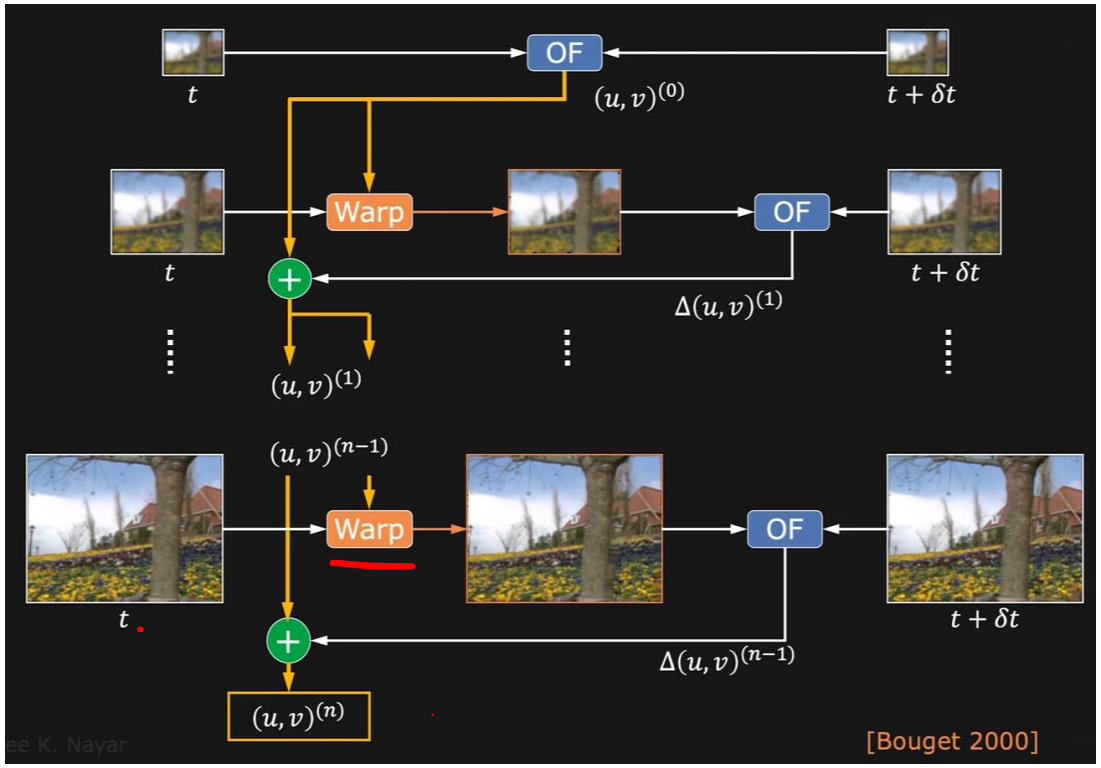
- Movimientos grandes (más de un píxel) – La aproximación por Taylor no es buena
  - Transiciones no lineales (aún siendo suaves) → *Refinamiento iterativo*
  - Saltos de intensidad (ya no locales) → Estimación gruesa a fina (*coarse-to-fine flow*)

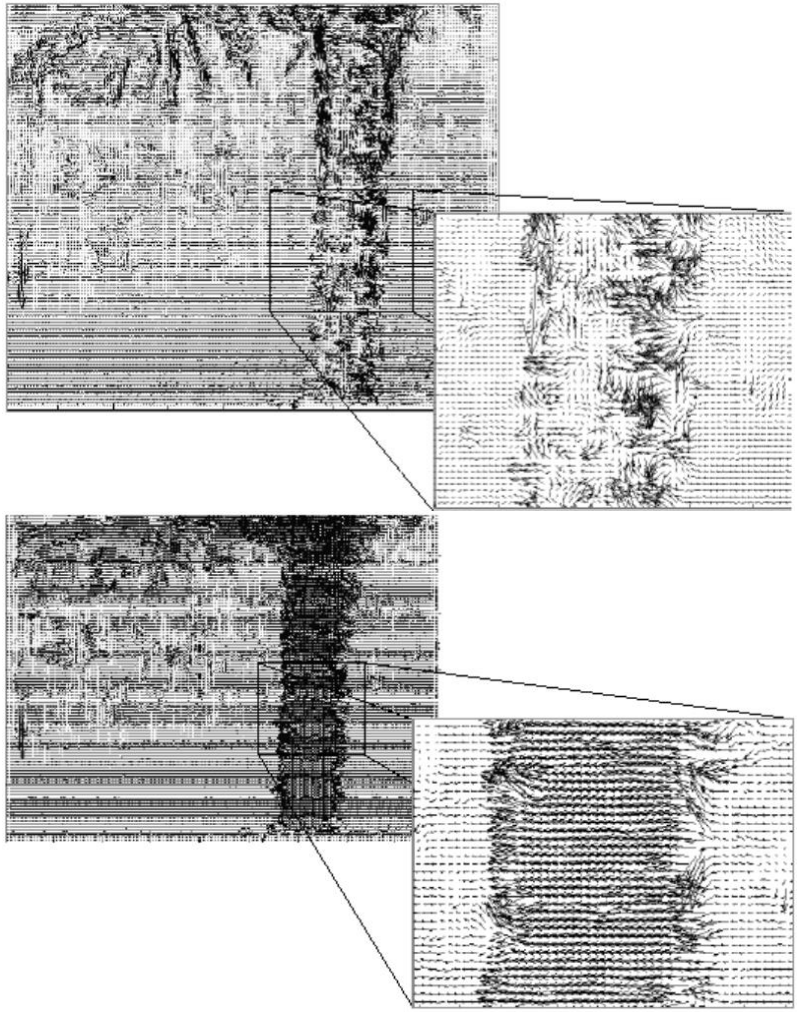
- Algoritmo iterativo de Lukas-Kanade
  1. Estimar la velocidad resolviendo las ecuaciones de Lucas-Kanade para cada píxel
  2. Deformamos la imagen del instante $I_t$ al instante $I_{t+1}$ usando el resultado del paso anterior. (Usando las técnicas de interpolación convencionales)
  3. Comparamos contra la verdadera imagen en $t + 1$. Volvemos a calcular el vector de desplazamiento y repetimos hasta converger.

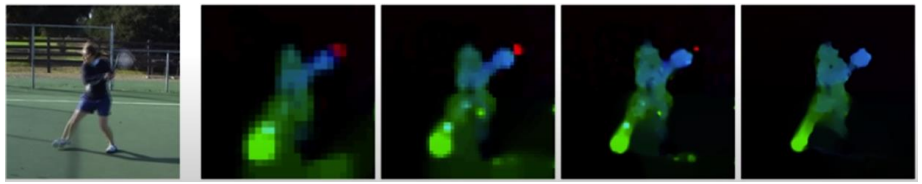
# LUCAS-KANADE POR JERARQUÍAS

- La manera de realizar esto es a través de pirámides gaussianas (deben ser gaussianas para evitar problemas de aliasing espacial)



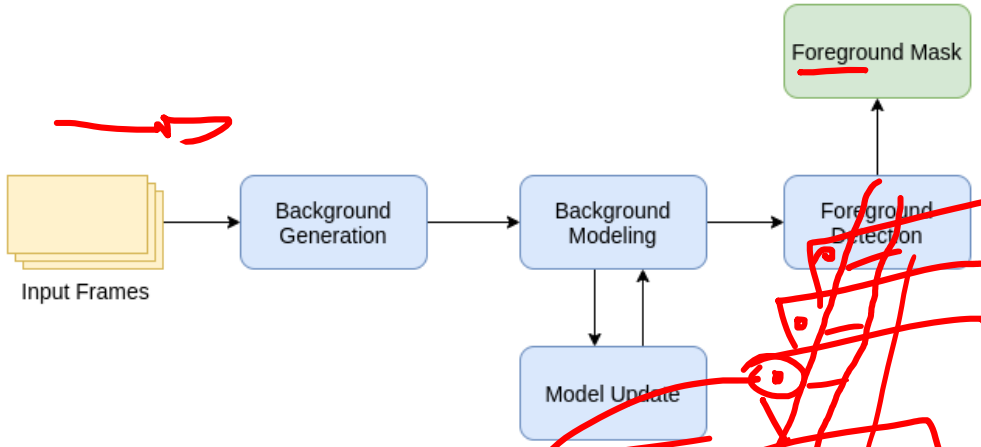




# LK POR JERARQUIAS - RESUMEN

- Sin pirámides: Falla en las regiones de grandes movimientos
- Con pirámides: Mejores resultados – Problemas en los bordes de la imagen (hay píxeles que aparecen y desaparecen)
- **LK-escaso (sparse):** Consiste en aplicar LK por jerarquías solo a los lugares donde hay buenas características para seguir (esquinas). OpenCV usaba LK-denso y actualmente usa LK-escaso.
- Aproximándonos a la actualidad (Brox et al, CVPR 2009) se utiliza el concepto de Lucas-Kanade pero con algunos agregados:
  - + Constancia de gradiente
  - + Region matching
  - + Minimización de energía con término de suavidad
  - + Keypoint matching (para grandes desplazamientos)
- Más en la actualidad todavía (Fisher et al, 2015)
  - FlowNet: Learning Optical Flow with Convolutional Networks

# SUSTRACCIÓN DE FONDO




- Detección de objetos en movimiento con cámaras estáticas.

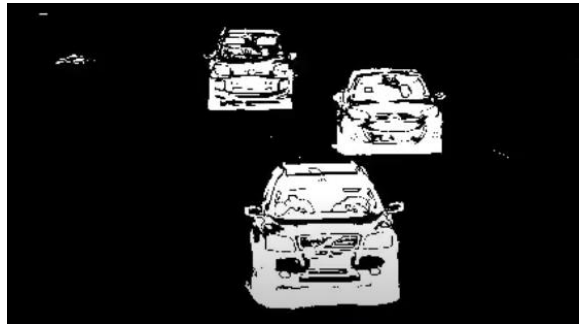
Enfoque "actual":

- La distribución de intensidad de brillo de los pixeles del fondo se modela como una mezcla de gaussianas
- Se "aprende" el fondo y se generan mascaras de segmentación para los objetos en movimiento
- "An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection". P. KaewTraKulPong and R. Bowden, 2001
- "Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation" . B. Godbehere, A. Matsukawa, K. Goldberg , 2012

Naive background substraction:

La mediana como estimador

- Se eligen N frames aleatorios y se calcula la mediana (background).
- Se resta el frame actual con la mediana y se binariza para obtener la mascara del objeto (*foreground*)
- Cada cierto intervalo se actualiza el modelo de *background* recalculando la mediana.



# TP:5

- **Objetivo:**
  - Implementar el detector de fondo naive usando la mediana como estimador. El algoritmo debe recibir el parámetro N (cantidad de frames utilizados para la estimación) y el intervalo de tiempo para recalcular el fondo.
  - Se deben generar las mascaras de foreground y aplicarlas a los frames para segmentar los objetos en movimiento.
  - Comparar con alguno de los métodos vistos en la practica basados en mezcla de gaussianas