Algoritmos de captura óptica de movimiento por substracción de video

Una implementación en sintaxis de Processing (Java)

Autor: Emiliano Causa

emiliano.causa@gmail.com , www. biopus.com.ar , www.emiliano-causa.com.ar

Palabras claves

Captura óptica de movimiento, substracción de video, visión artificial

Resumen

En este trabajo se plantearán los algoritmos necesarios para realizar diferentes tipos de captura óptica de movimiento por substracción de video. Este tipo de técnicas son de dominio público y no pretendo presentarme como creador de las mismas, sin embargo el objetivo de este trabajo es mostrar los algoritmos que realicé, originalmente, y que implican una resolución propia del asunto. Los ejemplos que se muestren de dichos algoritmos fueron programados por quién escribe en el lenguaje Processing (www.processing.org).

1. Captura de óptica de movimiento por substracción de video

En el campo de la captura de movimiento (*Motion Capture*) las técnicas de substracción de video, son una de las más sencillas. Esta técnica consiste en tomar una imagen de la escena sin movimiento, a la que llamaremos “fondo”, y a partir de ahí, restársela a cada nueva imagen capturada en la escena. Se realiza, en el flujo de video que proviene de la cámara, una substracción entre cada fotograma y el fotograma de referencia, el fondo (figura 1). Esta resta se realiza operando píxel por píxel, de una de estas imágenes, con sus respectivos píxeles de la otra imagen. Cuando los píxeles de una y otra imagen coinciden, la substracción entre valores idénticos resulta cero, lo que traducido a color es el negro (cero luz, o falta total de luz); y en el caso contrario aparecen otros tonos. De esta forma, es fácil descubrir los píxeles que cambiaron entre los dos fotogramas (el actual y el fondo), dado que adoptan tonos diferentes al negro y de ahí que los movimientos que se realicen frente a la cámara queden distinguidos de entre un fondo negro.

Este tipo de captura es muy sencilla, dado que no requiere que el sujeto del movimiento esté intervenido de ninguna forma; es decir, por ejemplo, si se quiere capturar el movimiento de una persona, no requiere que se lo coloque nada a la persona, ni trajes especiales, ni sensores. La captura se realiza por simple comparación de imágenes. Esto, que en principio es una ventaja, tiene como desventaja el hacer a esta técnica muy sensible a los cambios de iluminación. Dado que el criterio es considerar equivalentes el “cambio de imagen” a la “existencia de movimiento”, cualquier cambio en la imagen es automáticamente tomado por un movimiento.

Las dos fuentes más comunes de errores en este tipo de captura, son los cambios de iluminación y los movimientos de cámara. Por ejemplo, si se utiliza una técnica de este tipo, en un espacio donde ingresa luz natural, es probable que los cambios de iluminación de las diferentes horas del día generen problemas. Otro problema posible, es cuando la cámara sufre algún tipo de movimiento, por ejemplo, por un leve golpe o alguna vibración del piso. Cualquiera de estas dos cosas, podría hacer, en algunos casos, que la técnica deje de ser efectiva y capte movimiento donde no lo hay.

Otro de los problemas que existen, es que este sistema requiere un buen contraste en la imagen que obtiene la cámara, dado que es necesario aplicar un filtro para distinguir entre “cambios de imagen” y “ruido de la cámara o la iluminación”. Por esto, es un sistema que requiere un buen nivel de luz para funcionar correctamente. Esto a veces complica la situación a la hora de querer trabajar con captura de movimiento en lugares donde se realizan proyecciones de video y que requieren penumbra u oscuridad. Sin embargo esto se puede solucionar con la utilización de cámaras y luces infrarrojas.

Figura 1: captura del movimiento mediante substracción de video. La imagen de la izquierda es el fondo, la de la derecha es un fotograma de la escena actual y debajo el resultado de la substracción.

1.1. Substracción de imagen de referencia, versus substracción de fotogramas anteriores

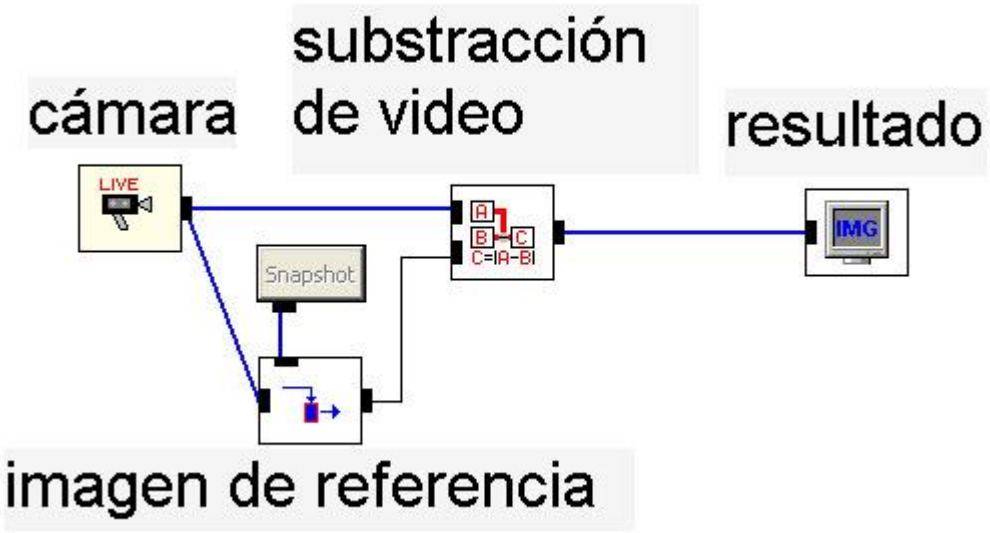
Existen dos posibles formas de realizar una captura de movimiento por substracción de video, la primera es la Substracción de una Imagen de Referencia (que llamaremos SIR) y la Substracción de Fotogramas Anteriores (que llamaremos SFA).

La SIR, como ya lo explicamos, consiste en tomar una primera imagen que sirve como referencia del "espacio sin movimiento" y contra la cual se realiza la operación de comparación (por substracción de video) con cada uno de los nuevos fotogramas.

La SFA, en cambio, consiste en comparar cada uno de los fotogramas con los anteriores, es decir, se usa un retardo (*delay*) que demora la imagen para luego realizar la substracción entre la imagen demorada y la actual.

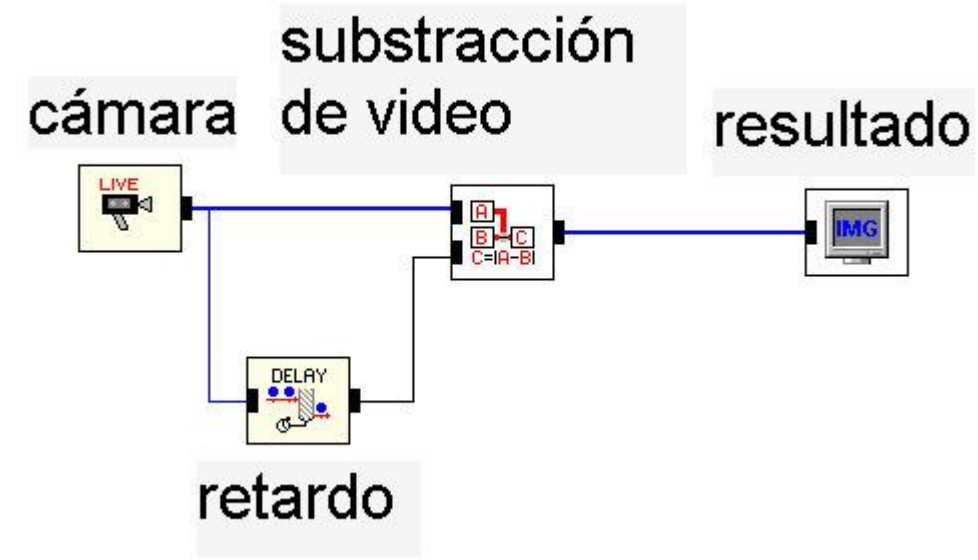
Para ser más ilustrativo voy a tomar dos ejemplos realizados con EyesWeb (www.eyesweb.org), un excelente software open-source, desarrollado por un equipo de investigación de la Universidad de Génova. Tomo estos ejemplos, por que EyesWeb es un lenguaje de programación visual (basado en objetos) y es sencillo ver el esquema del flujo de la información. Por ejemplo, la Figura 2, muestra un algoritmo SIR, de izquierda a derecha, el video corre desde la cámara hasta el *display*, pasando, por la parte baja, por un objeto que toma una foto (una imagen estática) de la imagen de referencia y luego por un objeto que realiza la substracción de ambas imágenes.

Figura 2: SIR: captura de movimiento por Substracción de Imagen de Referencia
Ejemplo realizado con EyesWeb (www.eyesweb.org)



En la Figura 3, se observa un algoritmo de SFA. En grandes rasgos es idéntico, salvo que el objeto de la “imagen de referencia” es reemplazado por un objeto de retardo que demora la imagen, por lo que al objeto de substracción llegan dos imágenes diferentes, una actual y otra del pasado.

Figura 3: SFA: captura de movimiento por Substracción de Fotogramas Anteriores.
Ejemplo realizado con EyesWeb (www.eyesweb.org)



Existen diferencias entre ambos métodos con respecto a la estabilidad y al tipo de parámetros que captura.

La diferencia, a nivel de ventajas y desventajas, entre ambos métodos son las siguientes

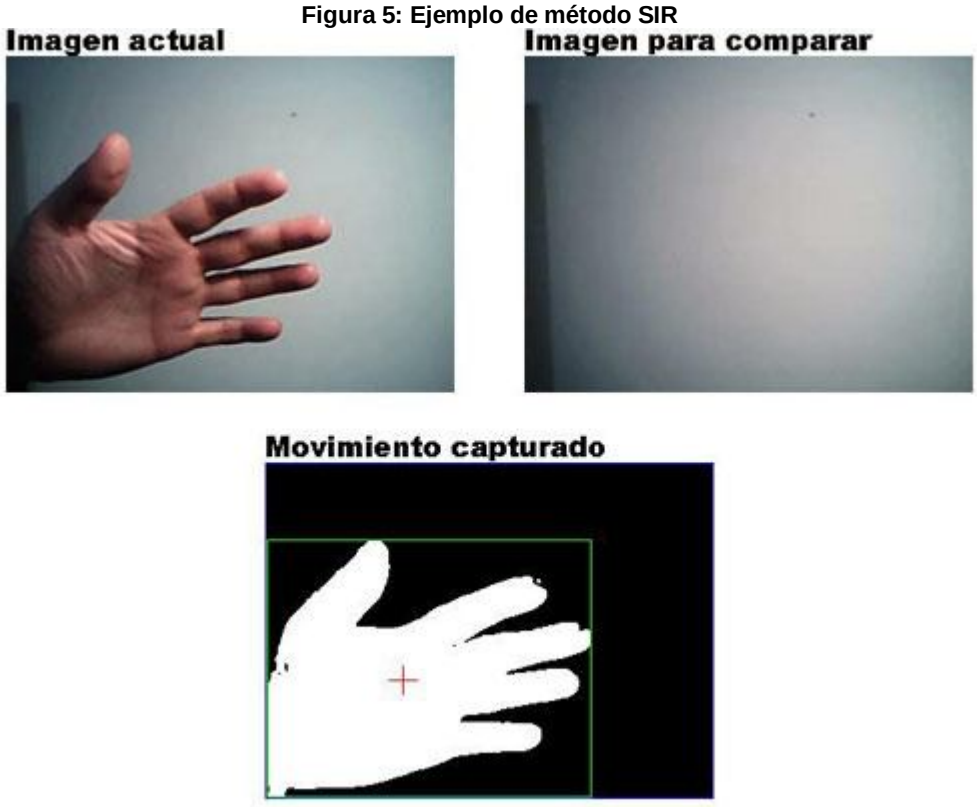
Figura 4: Cuadro de comparación entre métodos SIR y SFA.

SIR (Substracción de Imagen de Referencia)	SFA (Substracción de la Imagen Anterior)
Es muy sensible a las condiciones de luz y alteración de la imagen en general (como los movimientos de cámara).	Si bien en un principio responde igual que la SIR, rápidamente se estabiliza, lo que los hace un método bastante seguro.
Sirve para captar la silueta del objeto en movimiento.	No sirve para captar la silueta, sino la diferencia de movimiento entre fotogramas.
Si hay algún cambio de iluminación o movimiento de cámara, el método deja de funcionar.	Si el sujeto se queda quieto, el método es incapaz de captarlo.
Sirve para captar presencia	Sirve para captar cantidad de movimiento

Como muestra el cuadro de la Figura 4, el método SFA es mucho más estable, sin embargo, no sirve para captar presencia, dado que si el sujeto se queda inmóvil el método no lo detectará. Esto se debe a que coinciden el fotograma actual con el anterior (en ambos, el sujeto se encuentra en el mismo lugar). En cambio, el método SIR es mejor para detectar presencia, pero es muy inestable.

En la Figura 5 se puede ver un ejemplo de captura por SIR, en donde se ve la silueta de la mano.

Figura 5: Ejemplo de método SIR



En cambio en la figura 6, se observa como una mano en movimiento genera una captura de la diferencia entre las posiciones de la mano.

Figura 6: Ejemplo de método SFA

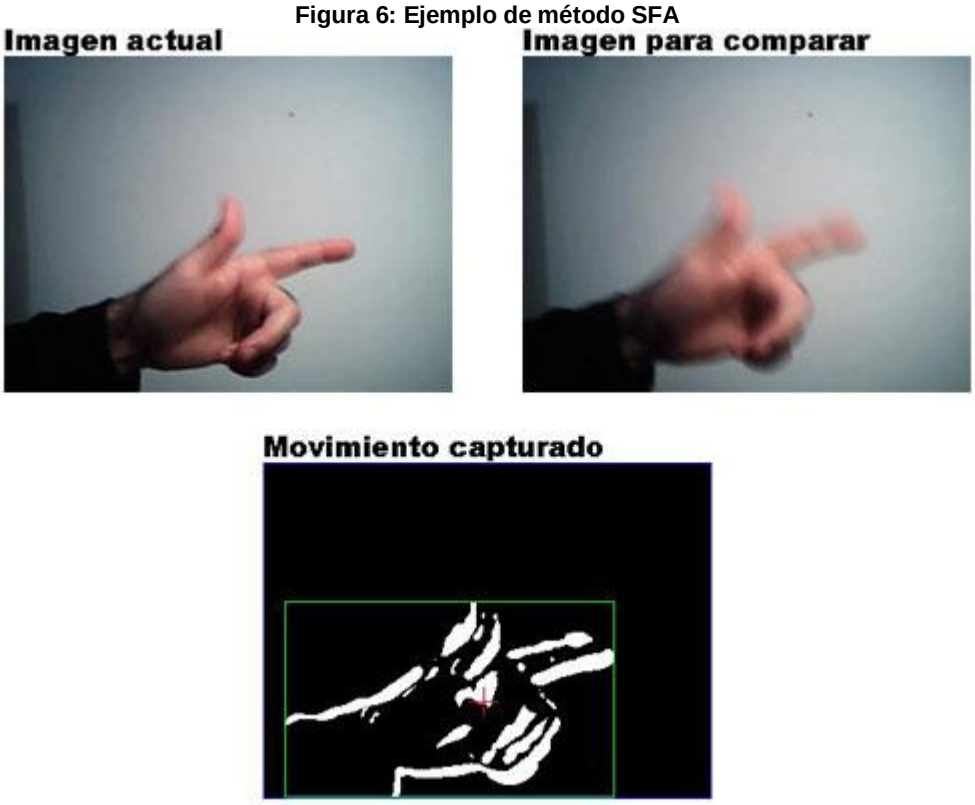


Figura 7: Ejemplo de método SIR, en donde se hizo una toma defectuosa de la imagen de referencia

Otro asunto importante en el sistema SIR es que la imagen de referencia debe ser tomada cuidadosamente, dado que si la imagen no corresponde enteramente al fondo (es decir a todo lo que estará netamente inmóvil en la escena), entonces el algoritmo interpretara como movimiento cosas que no lo son. Esto se puede ver en la mancha de la esquina superior izquierda de la figura 7.

2. Algoritmo de substracción

Ahora veremos cómo se desarrolla el algoritmo necesario para hacer la substracción de video. Este algoritmo realiza varias tareas:

1) Realiza la substracción entre los píxeles
2) Los promedia para filtrar el posible ruido generado por la cámara o la iluminación
3) Les aplica un umbral para decidir cuáles de ellos son movimiento
4) Obtiene parámetros del movimiento como:
    i) área
    ii) centro del área
    iii) márgenes del área de movimiento

Figura 8: Parámetros de movimento

Como se puede ver en la Figura 8, el área es la cantidad total de píxeles en blanco (es decir de píxeles con diferencia suficiente para ser considerados “movimiento”), el borde o límite son las mínimos y máximos valores horizontales y verticales que alcanzan dichos píxeles, y el centro, no es el centro de estos bordes, sino el centro de gravedad de esta área:

Debajo se encuentra el Algoritmo 1 (substracción de video), esta función recibe las dos imágenes como parámetro externo. Estas imágenes se llaman **actual** y **fondo.**

Algoritmo 1: substracción de video

```
void compararConFondo( Imagen actual , Imagen fondo ){

       //parte 1: inicia valores
    area=0;
    int totx=0;
    int toty=0;
    int maxx=0;
    int minx=ancho;
    int maxy=0;
    int miny=alto;

       // parte 2: obtiene diferencias absolutas
    for(int i=0;i<ancho;i++){
      for(int j=0;j<alto;j++){

        float valor;
```

```
      valor = Math.abs( readRed(actual,i,j) - readRed(fondo,i,j) );
      valor += Math.abs( readGreen(actual,i,j) - readGreen(fondo,i,j) );
      valor += Math.abs( readBlue(actual,i,j) -  readBlue(fondo,i,j) );
      dife[i][j] = valor;

    }
  }

     // parte 3: promedia valores entre pixeles vecinos
  for(int g=0;g<cantMedia;g++){

        float valor;
        for(int i=vecindad;i<ancho-vecindad;i++){
          for(int j=vecindad;j<alto-vecindad;j++){

            valor=0;
            for(int k=i-vecindad;k<=i+vecindad;k++){
              for(int l=j-vecindad;l<=j+vecindad;l++){
                valor += dife[k][l];
              }
            }
            dife[i][j] = valor / (vecindad*2+1) / (vecindad*2+1) ;
          }
        }
  }

     // parte 4: selecciona píxeles con movimiento en función del umbral
     // y define los márgenes y área de movimiento
  for(int i=1 ; i<ancho-1 ; i++){
    for(int j=1 ; j<alto-1 ; j++){

      if( dife[i][j] > umbral*3 ){
        datos[i][j] = 1;

        area ++;
        maxx = (i>maxx ? i : maxx);
        maxy = (j>maxy ? j : maxy);
        minx=  (i<minx ? i : minx);
        miny = (j<miny ? j : miny);
        totx += i;
        toty += j;

      }
      else{
        datos[i][j] = 0;
      }
    }
  }

     // parte 5: busca el centro promediando las posiciones de los márgenes
  if( area > 0 ){
    centx = int(totx/area);
    centy = int(toty/area);
  }

     // parte 6: obtiene los márgenes de la región
  if( area > 0 ){

    tMovi = int(miny);
    bMovi = int(maxy);
    lMovi = int(minx);
    rMovi = int(maxx);

    movAncho = abs(lMovi-rMovi);
    movAlto = abs(tMovi-bMovi);
  }
}
```

En la primer parte (comentada en el algoritmo como “//parte 1:”) se inician las variables necesarias para calcular los parámetros. Al finalizar el algoritmo, estas variables contendrán los valores de los parámetros.

2.1. Substracción de componentes con valor absoluto

En la segunda parte del Algoritmo 1, dos **ciclos for** son los encargados de recorrer todos los píxeles de las dos imágenes. El primer ciclo recorre a lo ancho y el segundo a lo alto. En cada paso de los ciclos, se calcula la diferencia entre los colores de ambas imágenes, esto es, se obtiene la diferencia por cada componente color (Red, Green y Blue) y se los suma. Esto mediante las funciones **readRed()**, **readGreen()** y **readBlue()**:

```
float valor;
valor = Math.abs( readRed(actual,i,j) - readRed(fondo,i,j) );
valor += Math.abs( readGreen(actual,i,j) - readGreen(fondo,i,j) );
valor += Math.abs( readBlue(actual,i,j) -  readBlue(fondo,i,j) )
dife[i][j] = valor;
```

Una aspecto importante de este cálculo, es el uso del valor absoluto (**Math.abs( )**). Esta función se utiliza a los fines de evitar obtener valores negativos en los cálculos, dado que sólo nos interesa el módulo de la diferencia (es decir la distancia entre los valores). Por ejemplo, si un píxel de la imagen actual tiene un color celeste (cuyos valores de componentes son Red = 100 , Green = 120 , Blue = 255) y el píxel respectivo del fondo posee un rojo (Red = 255 , Green = 0 , Blue = 0) entonces la substracción resultaría:

Valor = abs(100-255) + abs(120-0) + abs(255-0) = 155 + 120 + 255 = 530

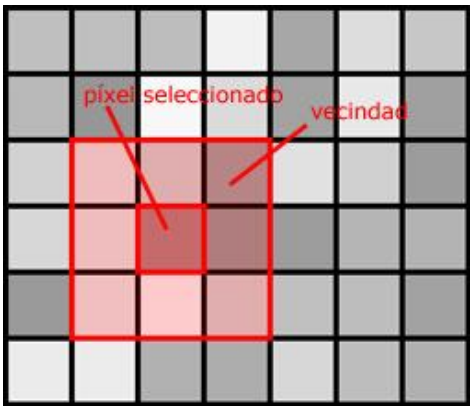
Un detalle a tener en cuenta, es que este resultado puede adoptar valores entre 0 y 756 (255*3). Si bien se podría dividir por 3 para normalizar los valores entre 0 y 255, esto requiere la repetición de una operación más en todo el ciclo y es innecesario. Luego habrá que tomar la precaución de multiplicar el umbral por 3 en la cuarta parte del algoritmo.

2.2. Promedio de las diferencias de la vecindad

En la tercer parte del Algoritmo 1, se recorre la matriz de diferencias, generada en el paso anterior, (**dife[][]**) y se promedia cada píxel con su vecindad. Esto se hace para minimizar el ruido. Los píxeles aislados que poseen diferencias importantes, responden más a la existencia de ruido que a la captación de una diferencia real, por eso con este método se asegura de que no existen píxeles aislados con grandes valores.

En la Figura 9 se muestra un Píxel y su vecindad, conformada por todos los píxeles contiguos a este, así, de cada píxel se recorre toda su vecindad y se obtiene un promedio que es cargado en el píxel.

Figura 9: promedio de la vecindad



Para realizar el promedio con la vecindad se usan 5 **ciclos for**. El primero controla la cantidad de veces que se realiza esta operación, el segundo y tercero recorren la matriz completa, y el cuarto y

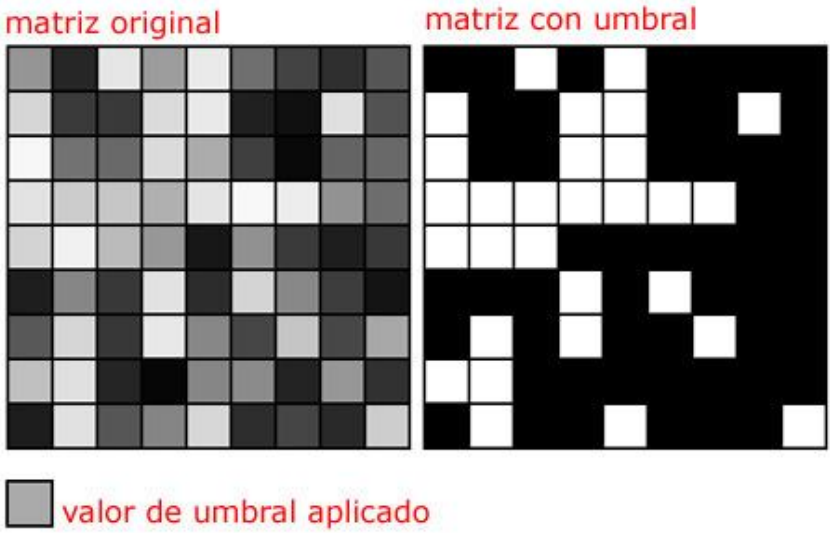
quinto, se encargan de recorre submatrices para cada píxel (es decir, recorren la vecindad del píxel). El cálculo del promedio con la vecindad se realiza en la siguiente sección:

```
valor=0;
for(int k=i-vecindad;k<=i+vecindad;k++){
   for(int l=j-vecindad;l<=j+vecindad;l++){
     valor += dife[k][l];
   }
 }
 dife[i][j] = valor / (vecindad*2+1) / (vecindad*2+1) ;
```

2.3. Aplicación de un umbral

En la cuarta parte del Algoritmo 1, se recorre píxel por píxel la matriz de diferencias, ya promediada, y se seleccionan los que superan un umbral. En la Figura 10 se muestra un ejemplo de aplicación de umbral, en donde todo píxel que supere un valor determinado (en la figura se muestra el gris equivalente al valor de umbral aplicado) se le asigna el máximo valor (en este caso el blanco) y a los que no, el negro.

Figura 10: aplicación de umbral



Esta parte de l algoritmo también se encarga de contabilizar los píxeles para el **área** y calcular los **márgenes o bordes** que circunscriben esta región:

```
if( dife[i][j] > umbral*3 ){
   datos[i][j] = 1;

   area ++;
   maxx = (i>maxx ? i : maxx);
   maxy = (j>maxy ? j : maxy);
   minx=  (i<minx ? i : minx);
   miny = (j<miny ? j : miny);
   totx += i;
   toty += j;

 }
 else{
   datos[i][j] = 0;
 }
```

En el condicional, el umbral se multiplica por 3 debido a que cuando se sumaron las diferencias de los 3 componentes color, no fueron promediados, por lo que los valores podían estar entre 0 y 765, a

fines de normalizar conviene que el umbral sea definido como una valor ente 0 y 255, por lo que es necesario multiplicarlo por 3 dentro del algoritmo.

En esta etapa, se calcula el **área** (la superficie) del movimiento, esto se hace a mediante la contabilización de los píxeles que superan el umbral, como se puede ver en la tercer línea del ejemplo anterior (**area++**).

Otro de los datos que se obtienen son los **bordes** o **márgenes** de dicha área, lo que se consigue actualizando los mínimos y máximos de las posiciones horizontales y verticales de los píxeles en el área:

```
maxx = (i>maxx ? i : maxx);
maxy = (j>maxy ? j : maxy);
minx=  (i<minx ? i : minx);
miny = (j<miny ? j : miny);
```

El **centro de gravedad** del área se obtiene mediante un promedio de las posiciones horizontales y verticales de los píxeles. Esto se hace a través de la acumulaciones de estos valores en las variables **totx** y **toty** que luego serán divididos por la cantidad de píxeles (el **área**) en la parte 5 del Algoritmo:

```
if( dife[i][j] > umbral*3 ){
      datos[i][j] = 1;

      area ++;
      …
      totx += i;
      toty += j;
…
}

if( area > 0 ){
  centx = int(totx/area);
  centy = int(toty/area);
}
```

Por último, los píxeles que pertenecen al movimiento se registran en una matriz llamada **datos[ ][ ]**, en la que el movimiento se nota como 1 y lo inmóvil como 0. Esta matriz queda disponible para posteriores procesos que requieran hacer otras funciones u objetos.

En la sexta parte, se establecen los nuevos valores de los parámetros:

```
if( area > 0 ){
    tMovi = int(miny);
    bMovi = int(maxy);
    lMovi = int(minx);
    rMovi = int(maxx);

    movAncho = abs(lMovi-rMovi);
    movAlto = abs(tMovi-bMovi);
}
```

2.4. Los valores de umbral

En este punto podemos preguntarnos cuál es un valor adecuado para el umbral. No existe un valor fijo que cumpla con dicha condición. El umbral depende necesariamente de las condiciones de luz con las que se esté trabajando, dado que la función del umbral es filtrar el ruido. Es decir, si las condiciones de luz son muy buenas, la cámara minimizará el ruido, estará trabajando en condiciones óptimas y la relación señal/ruido será muy buena, por lo que un valor de umbral bajo alcanzará. Por el contrario, si las condiciones de luz son malas y la cámara por ende trabajará con tiempos de exposición prolongados y con un nivel de ruido muy alto, entonces será necesario utilizar un umbral alto para filtrar el ruido.

La Figura 11 muestra como funciona el umbral, en el cuadrado de la parte superior se muestra el valor de gris correspondiente al valor de umbral. En la figura de la parte derecha (**filtrado**) se puede observar como todos los píxeles de la matriz **original**, que son más oscuros, que este umbral, pasaron a ser negros, y los que son más claros, pasaron a ser blancos.

Si observamos detalladamente ambas figuras puede verse como la forma que se insinúa en la matriz **original** se define totalmente en la matriz **filtrada**.

En la Figura 12, se utiliza el mismo criterio, pero en este caso se bajó nivel del umbral, lo que produce que algunos píxeles que son parte del fondo y que tienen ruido alto, pasaron a ser parte de la figura de la matriz. Es decir algunos de los píxeles que debían transformarse en negro, quedaron al final como blanco.

En la Figura 13, sucede lo contrario, es decir, el valor de umbral que se utiliza es alto, lo que ocasiona que algunos píxeles que son parte de la figura, pasaron a ser fondo. O dicho de otra forma, algunos de los píxeles que debían quedar en blanco, quedaron negros. El ruido no pasa el filtro, pero el filtro es tan alto que parte de la figura no sobrevive al filtro.

En las Figuras 14 y 15, se muestran ejemplos de filtros inadecuados pero aplicados a casos reales de captación de movimiento. En ambos casos, en uno por exceso y en otro por defecto, se pueden ver como el sujeto de la captura (una mano) es mal interpretada por el filtro.

Figura 11: filtro por umbral con valor adecuado

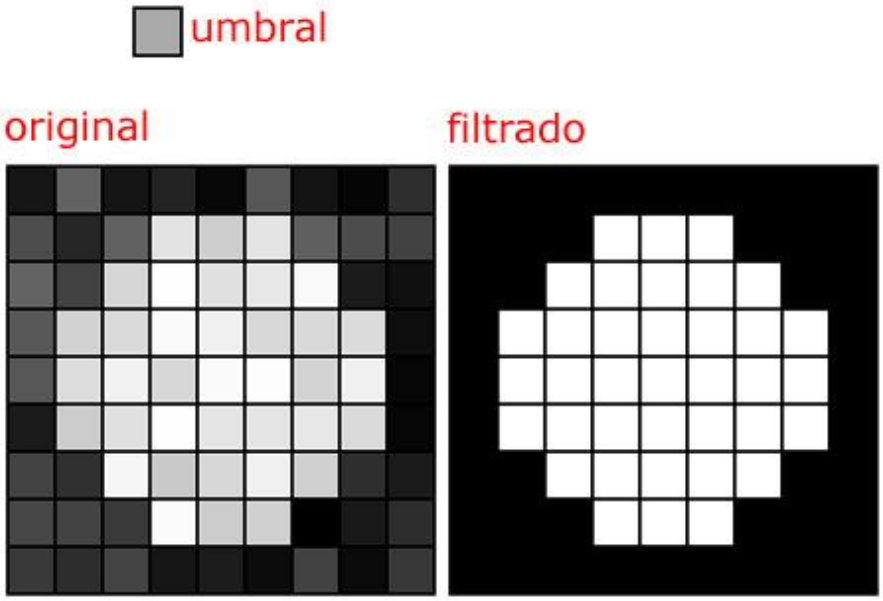


Figura 12: filtro por umbral con valor bajo

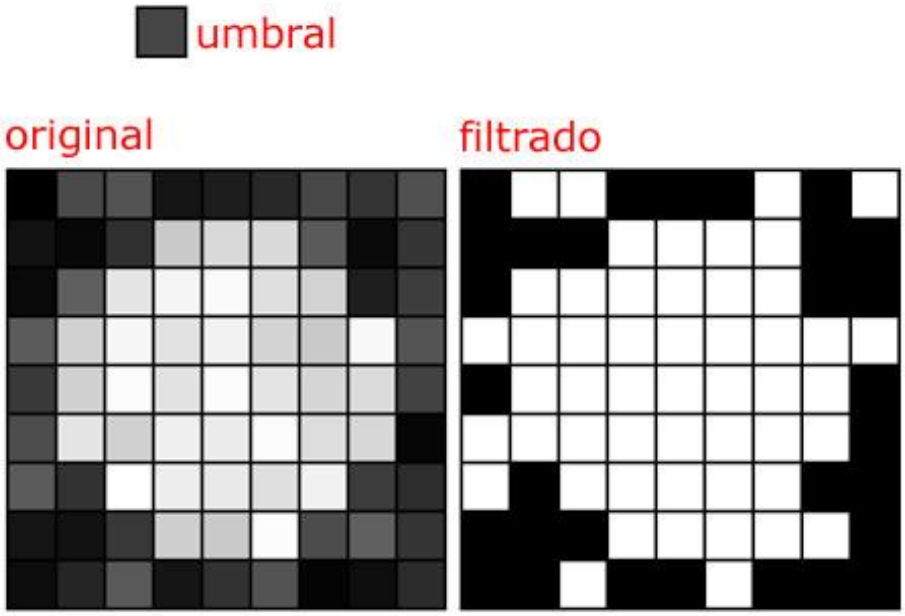


Figura 13: filtro por umbral con valor alto

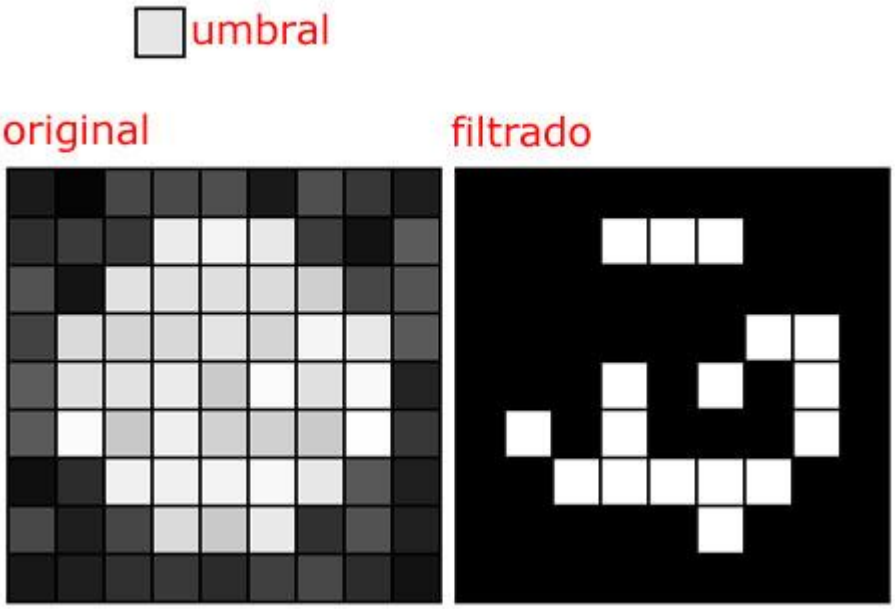


Figura 14: filtro por umbral con valor bajo

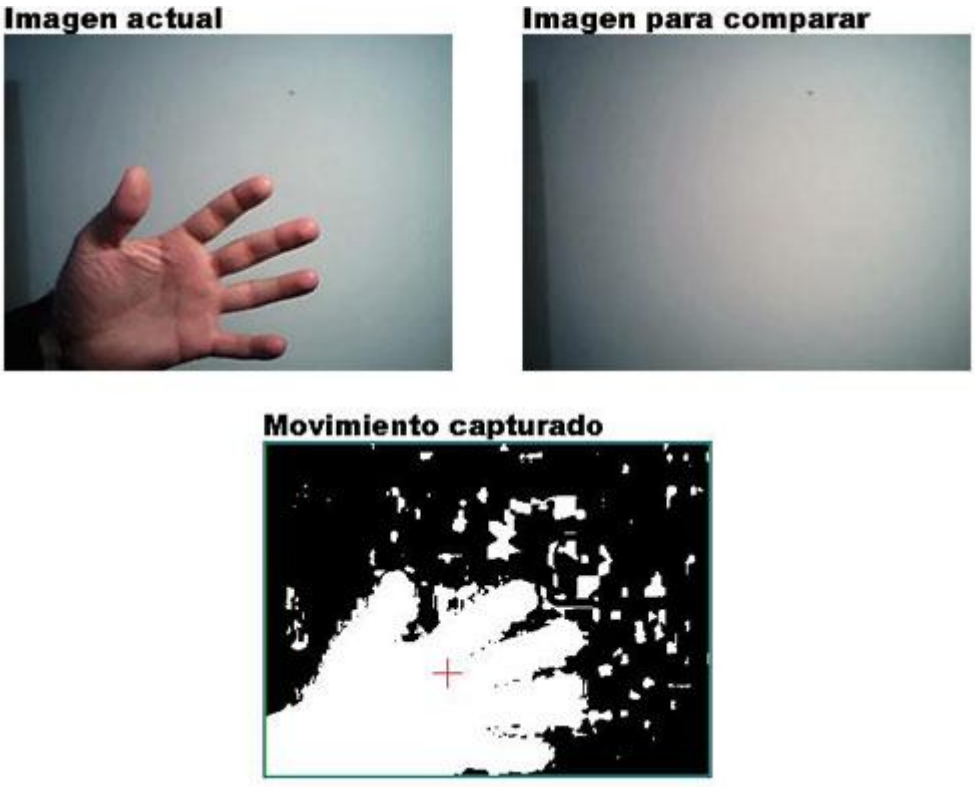
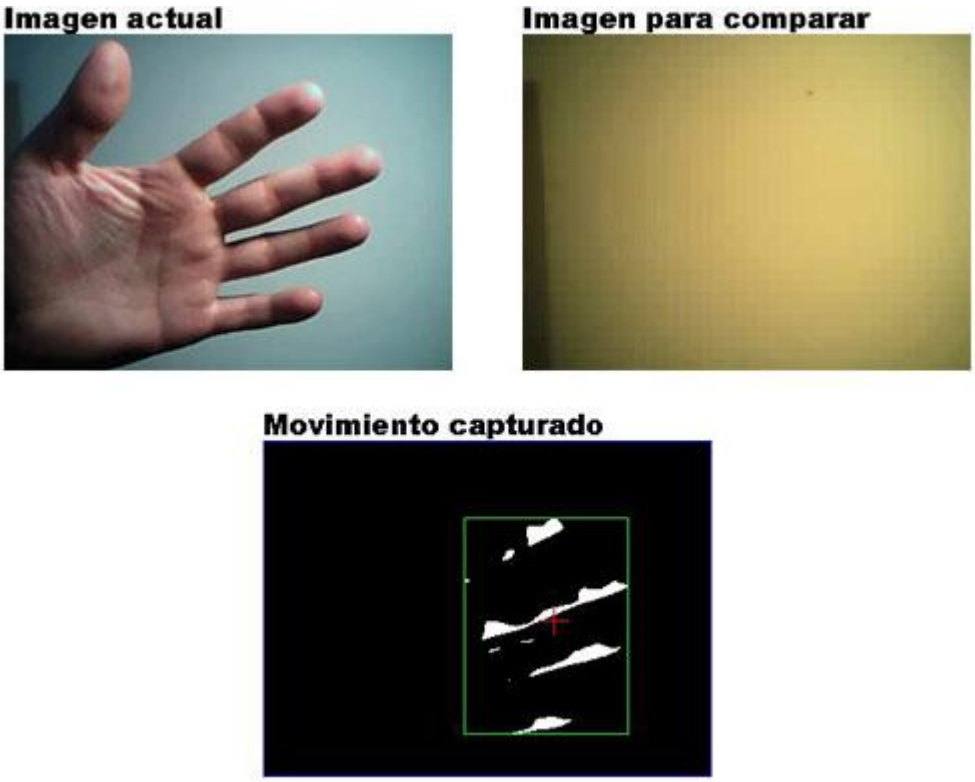


Figura 15: filtro por umbral con valor alto



3. Captura de movimiento en regiones del cuadro

Muchas veces es necesario capturar el movimiento dentro de una región determinada del cuadro tomado por la cámara. A esos fines desarrollé un sencillo algoritmo que permite conocer el porcentaje de la región que esta cubierto por movimiento.

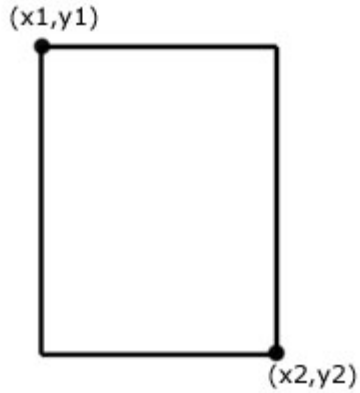
La Figura 16 muestra una captura de movimiento dentro de una región del cuadro. Por supuesto, este algoritmo está limitado a la toma de regiones estrictamente rectangulares.

Algoritmo 2: captura de movimiento en región

```
float movEnArea(int x1, int y1, int x2, int y2){

    int total=0;

    for(int i=x1;i<=x2;i++){
      for(int j=y1;j<=y2;j++){
        if(datos[i][j] > 0){
          total++;
        }
      }
    }

    return float(total)/((x2-x1+1)*(y2-y1+1));
  }
```

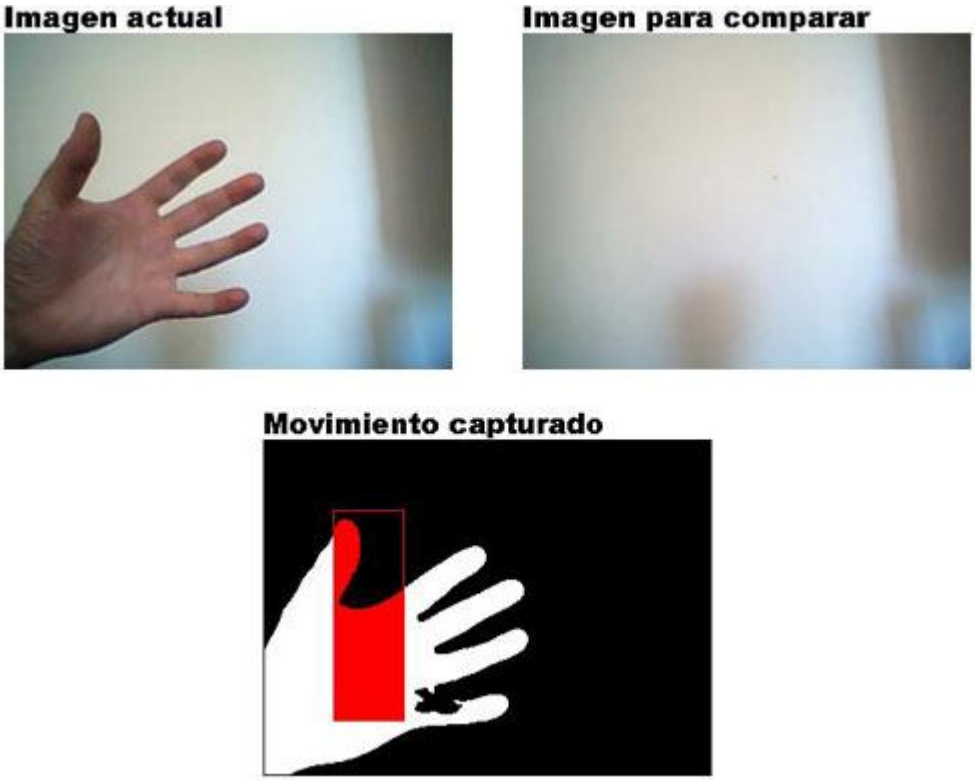
Una vez que el algoritmo de substracción de video, calculó las diferencias y aplicó el umbral, los datos de movimiento quedan en una matriz llamada **datos[ ][ ]**, la cual posee celdas con valores en 1 y en 0 para representar movimiento y fondo respectivamente.

Delimitamos la región que queremos explorar con un rectángulo definido por dos puntos (que son sus esquinas opuestas por la diagonal), es decir que lo definimos por 4 coordenadas, **x1, y1, x2, y2**:



De esta forma el algoritmo utiliza estas coordenadas como límites de los **ciclos for** que recorren la región. En esta exploración píxel por píxel delimitada por este rectángulo, se contabilizan los píxeles con movimiento usando la variable **total**. Luego, se divide el valor de dicha variable por la cantidad total de píxeles explorados, de esta forma el resultado siempre está dentro del rango entre cero y uno, sin importar el tamaño del rectángulo.

Figura 16: captura de movimiento en una región del cuadro



4. Los límites de este tipo de captura de movimiento

Este tipo de captura de movimiento es sencilla de implementar, y muy útil a la hora de medir el movimiento de un único sujeto, siempre y cuando no se quiera interpretar una diferenciación de partes. Es decir, si quiero captar la presencia o no de una persona, o captar el movimiento de esta persona, este tipo de algoritmo será suficiente, pero si deseo captar varias personas por separado o interpretar las distintas partes del cuerpo de una persona en movimiento, entonces este algoritmo no alcanzará.

En la figura 17, se puede ver la captura de movimiento de dos objetos que se mueven en forma independiente (en este casos mis brazos). En la imagen se puede ver que los parámetros no son del todo verosímiles al fenómeno real. Por ejemplo, los **márgenes** encierran a ambos brazos y por ende si estos se separan, abrirán estos **márgenes** aunque el **área** no se modifique. Más notorio es aún el hecho de que el centro de gravedad parece no representar a ninguno de ambos brazos.

Figura 17: captura de movimiento de dos objetos independientes

Para resolver estos problemas será necesario aplicar algoritmos de Análisis de Formas para interpretar manchas conexas e inconexas, y así discriminar la dependencia e independencia entre objetos en movimiento. Este tema no será tratado aquí, sino que será resuelto en un futuro escrito.

5. Conclusión

La captura óptica de movimiento por substracción de video es una técnica muy sencilla y bastante potente, el costo de su sencillez se paga con las complicaciones que surgen de su sensibilidad a los cambios de iluminación. Sin embargo esta aparente desventaja también a permitido el uso de la luz como forma de interactuar, ya sea mediante el uso de la sombra, como en los trabajos de Mine Control (www.mine-control.com) o de Scout Snibbe (snibbe.com), o con el uso de linternas, también en Mine-Control.

Faltó tratar en este trabajo las diferentes posibilidades de puestas en escena de esta técnica, ya sea con luz normal o infrarroja, esta última brinda grandes posibilidades para el trabajo con proyecciones, como es el casos de la obra *The Jew of Malta* de Art+Com (www.artcom.de).

Cabe destacar también el trabajo de Nicolas Boillot (www.fluate.net) quien se dedica a realizar este tipo de captura de movimientos pero sobre la imagen en vivo de la televisión, obteniendo resultados muy interesantes y demostrando que siempre se puede utilizar una herramienta de una forma novedosa.

La captura óptica de movimiento por substracción de video nos ofrece una forma sencilla de vincular, interactivamente, el cuerpo del público en nuestros trabajos. Posee la inmensa ventaja de no tener que intervenir el cuerpo de las personas de ninguna forma y de no depender de más dispositivos físicos que de una cámara. En nuestro país, una cámara USB sencilla no cuesta más de 40 dólares, lo que no es un dato menor a la hora de trabajar con u este tipos de tecnologías en el arte.

Creo que como hizo Nicollas Boillot, tenemos la posibilidad de explorar nuevas potencialidades de estos algoritmos. En algunos trabajos, logré vincular el resultado de este tipo de algoritmos con Autómatas Celulares y los hallazgos son muy interesantes y abren toda una vía de investigación.

Emiliano Causa

Diciembre de 2007

6. Referencias bibliográficas

[1] opencvlibrary.sourceforge.net

[2] www.processing.org

[3] www.eyesweb.org

[4] www.mine-control.com

[5] snibbe.com

[6] www.artcom.de

[7] www.fluate.net

Todas consultadas el 9/Dic/2007 y actualmente en línea.