UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

PROYECTO SISTEMAS INFORMÁTICOS

Interacción persona-computador basada en el reconocimiento visual de manos

---

Arranz Aranda, Francisco

Liu Yin, Qi

López Cámara, Jose M.

Dirigido por:

Martin de la Calle, Pedro J.

Curso 2011 - 2012

Los alumnos abajo firmantes autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

En Madrid, a 27 de Junio de 2012.

Francisco Arranz Aranda Qi Liu Yin Jose M. López Cámara

Resumen

En este documento proponemos un método de interacción basado en las posiciones y los movimientos de la mano usando como única herramienta una cámara de video RGB. Esto se consigue reconociendo un conjunto de características genéricas que configuran la mano, a partir de las cuáles se identifican un amplio catálogo de gestos a los que se les podrá asignar cualquier evento para realizar acciones en un computador. Para su implementación, se emplean distintas técnicas de visión e inteligencia artificial entre las que cabe destacar el método de aprendizaje supervisado de *Máquinas de Soporte Vectorial* y el cálculo de contornos basados en gradientes de intensidad usando derivadas de *Sobel*.

Palabras clave: contorno, dedo, detección, gradiente, histograma, imagen, interacción, mano, OpenCV, seguimiento.

Abstract

In this paper, we propose an interaction method based on hand positions and movements using just a RGB video camera as tool. This target is reached by recognising a set of generic features of the hand from which it is identified a wide variety of gestures and assigning an event to each of these gestures to make actions on the computer. For its implementation, different artificial intelligence and vision technics are used such as the Support Vector Machine supervised learning method or the contours calculation based on oriented gradients using Sobel derivatives.

Key words: contour, detection, finger, gradient, hand, histogram, image, interaction, OpenCV, tracking.

Contenido

1. Introducción

En la actualidad existe una marcada tendencia a utilizar sistemas de reconocimiento gestuales que permitan la interacción de los usuarios con los dispositivos digitales.

Un ejemplo claro es la guerra existente entre las grandes marcas en el mercado del entretenimiento, por desarrollar este tipo de sistemas consiguiendo que los dispositivos se adapten de un modo natural a los gestos de los usuarios.

Siguiendo esta corriente, nos encontramos con la tecnología Kinect™ (Microsoft®), que se basa en el uso de varios sensores para realizar esta tarea: un sensor de color y un sensor de profundidad.

Con esta motivación y las premisas de desarrollar un sistema económico, accesible a todo el mundo y en tiempo real, el objetivo de este proyecto es el de desarrollar un sistema de interacción con el usuario a través de un sensor de color RGB (webcam) mediante la realización de gestos con la mano.

La economía y accesibilidad se consiguen debido a que no es necesaria la utilización de un hardware adicional aparte de una cámara RGB, la mayoría de las veces ya integrado en el sistema, y al coste cero de usar de librerías de código abierto (OpenCV[1], SVM^{light} [2]).

El uso en tiempo real se consigue mediante la implementación de un algoritmo eficiente y adaptado a la capacidad de procesamiento de las computadoras actuales.

Con todos estos requerimientos, se consigue un modo de interacción natural e intuitivo con las ventajas de una pantalla táctil pero sin la necesidad de un contacto directo con el dispositivo.

1.1. Uso de la API de OpenCV

En este proyecto se ha utilizado la API de OpenCV para gran parte de sus fases. *OpenCV* (*Open Source Computer Vision*) es una librería de código abierto que usa licencia BSD. Ésta incluye centenares de algoritmos de visión artificial.

Hay dos versiones estables: OpenCV 1.x implementada en C y OpenCV 2.x implementada en C++. Para el desarrollo de este proyecto se ha utilizado la segunda.

Con el objetivo de aportar una breve descripción de la potencialidad de esta librería, se describen a continuación sus módulos principales:

- **Core:** define las estructuras básicas, incluyendo arrays multidimensionales para la representación de imágenes, y las funciones básicas utilizadas por los otros módulos.
- **Imgproc:** módulo de procesamiento de imágenes que incluye filtros lineares y no lineares, transformaciones geométricas, conversiones entre espacios de color, histogramas y más.
- **Video**: módulo de análisis de vídeo que incluye estimación de movimiento, eliminación de fondo y algoritmos de seguimiento de objetos.
- **Calib3d**: módulo de análisis en 3D. Incluye calibración de cámaras simples y de visión estereoscópica, estimación de la posición de objetos, algoritmos de correspondencia estereoscópica y elementos de reconstrucción en 3D.
- **Features2d**: módulo de inteligencia artificial que incluye la creación de detectores de características principales, descriptores y encaje de patrones.
- **Objdetect**: módulo de reconocimiento basado en el anterior. Permite el reconocimiento de objetos y de instancias de clase definidas (por ejemplo, caras, ojos , personas, etc)
- **Highgui**: módulo de interfaz con el usuario (UI). Proporciona una sencilla interfaz para capturas de vídeo, imágenes y códecs de vídeo, además de sencillas herramientas de la UI.
- **Gpu**: módulo de aceleración gracias al uso del procesador gráfico.

Como se puede apreciar en la descripción de los módulos, aunque si que hay funciones generales que facilitan las cuestiones de reconocimiento, el reconocimiento de manos no está desarrollado debido a su complejidad.

2. Descripción del método

El proceso llevado a cabo para conseguir la interacción del usuario a través de la mano con el computador se basa en el procesamiento de una secuencia de imágenes (*frames*) obtenidas a partir de la cámara de video. Este proceso se divide en cuatro etapas principales: detección inicial de la posición de la mano, obtención de las características descriptivas de la misma, su seguimiento, e interpretación de la información obtenida en las etapas anteriores.

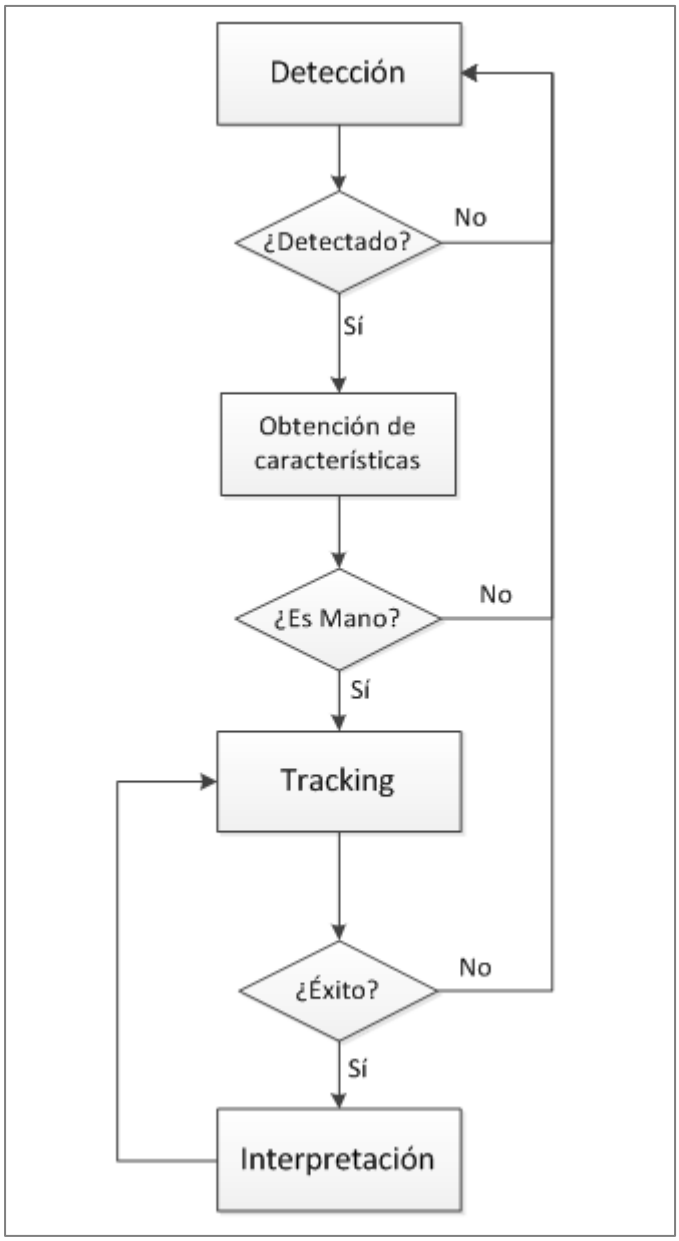

Ilustración 2.1 Diagrama de flujo del algoritmo seguido

La etapa de *detección* parte de una imagen tomada por la cámara y devuelve las regiones en las cuales se ha reconocido un patrón de mano. Esto se consigue calculando previamente un modelo, siguiendo un proceso de aprendizaje basado en una colección de imágenes, y buscándolo a lo largo de la imagen.

En el momento en el que la detección es satisfactoria, es decir, se dispone de una región, se prosigue con la etapa de *obtención de características*. En ésta se calcula el contorno de la mano y, a partir de éste, se obtienen los puntos estructurales de la misma. Sobre estos puntos se verifica que se cumplan unos ciertos criterios para comprobar que la región se corresponde realmente con una mano. Esta etapa se repite a lo largo de un cierto número de frames para asegurar que sea estable.

En el caso de que esta comprobación sea positiva y se haya alcanzado la estabilidad, se procede con el *seguimiento* de los puntos estructurales calculados en la etapa anterior a lo largo de los siguientes frames obtenidos desde la cámara.

Por último, y tras comprobar que el seguimiento ha sido correcto, se procede a la *interpretación* de la posición de los puntos, lo cual permitirá reconocer gestos y realizar acciones cuando éstos se produzcan.

Para cada nuevo frame se realizan las etapas de seguimiento e interpretación, hasta que un movimiento de la mano no sea reconocido, en cuyo caso el algoritmo se reiniciará volviendo a la etapa de detección.

3. Detección

Para realizar esta primera fase del algoritmo primero se debe conseguir un detector que identifique las manos en una imagen. El método aplicado se basa en el método usado por Dalal y Triggs[3] para la detección de personas. Para ello, se recoge información de una gran colección de imágenes tanto positivas (aparecen manos) como negativas (no aparecen manos) y se entrena un detector usando una técnica de aprendizaje denominada *Máquina de Vectores de Soporte* (del inglés *Support Vector Machine - SVM*). La información que se calcula de cada imagen es un *Descriptor de Histogramas de Gradientes Orientados* (del inglés *Histogram of Oriented Gradients Descriptor – HOG Descriptor*). Una vez obtenido el detector se podrá aplicar a imágenes para identificar manos en ellas. Para explicar lo que es un descriptor de HOG se deberá comprender antes qué es el gradiente de un píxel y, entonces, se podrá explicar que son los Histogramas de Gradientes Orientados y los descriptores de HOG.

3.1 Gradiente de un píxel

En matemáticas, el gradiente $\nabla\phi$ de un campo escalar ϕ en un punto x es un vector que indica la dirección en la cual el campo varía más rápidamente y su módulo representa el ritmo de variación de este campo ϕ en esa dirección. Para realizar su cálculo, dada una función $f(x,y,z)$ representada en un sistema de coordenadas cartesianas, su gradiente ∇f viene dado por la suma de las derivadas primeras de cada componente vectorial:

$$\nabla f = \frac{\partial f}{\partial x}\hat{x} + \frac{\partial f}{\partial y}\hat{y} + \frac{\partial f}{\partial z}\hat{z}$$

En el campo del tratamiento de imágenes digitales, el gradiente del píxel de una imagen tiene un significado similar: es un vector que indica la dirección en la cual se produce un mayor cambio en la intensidad o color de la imagen (en imágenes en escala de grises el gradiente se dirige a píxeles de menor valor, i.e. de píxeles blancos a píxeles más negros); y su módulo indica la magnitud de este cambio. Su cálculo se realiza de la misma manera pero, dada la naturaleza bidimensional de las imágenes, podemos eliminar la tercera componente:

$$\nabla f = \frac{\partial f}{\partial x}\hat{x} + \frac{\partial f}{\partial y}\hat{y}$$

Se puede representar el gradiente de un píxel en coordenadas polares por medio de un ángulo θ y una magnitud del gradiente $|G|$, las cuales resultan más útiles en la práctica:

$$\theta = atan2\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right)$$

$$|G| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Pero, a partir de una imagen, el cálculo de las derivadas primeras en x e y no se puede realizar de la misma forma que se haría en cálculo matemático ya que no se dispone de ninguna función f. En vez de ello, se realiza una aproximación de las mismas por medio del operador de Sobel con el cual, utilizando dos kernels (una para el eje x y otra para el eje y), se realiza una convolución sobre toda la imagen obteniendo dos imágenes que contienen el valor de la derivada de cada píxel. Entonces, dados una imagen A de tamaño WxH de la forma

$$A = \begin{pmatrix} a_{00} & a_{10} & \cdots & a_{(W-1)0} \\ a_{01} & a_{11} & \cdots & a_{(W-1)1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{0(H-1)} & a_{1(H-1)} & \cdots & a_{(W-1)(H-1)} \end{pmatrix}$$

donde a_{ij} es el valor de un píxel, y un kernel K de tamaño NxN, siendo N impar,

$$K = \begin{bmatrix} k_{00} & k_{10} & \cdots & k_{(N-1)0} \\ k_{01} & k_{11} & \cdots & k_{(N-1)1} \\ \vdots & \vdots & \ddots & \vdots \\ k_{0(N-1)} & k_{1(N-1)} & \cdots & k_{(N-1)(N-1)} \end{bmatrix}$$

una convolución sobre la imagen A es la aplicación del kernel K a cada uno de los píxeles de ésta, de forma que el valor de píxeles de la imagen resultante A' de tamaño WxH son calculados siguiendo la expresión:

$$a'_{ij} = \sum_{r=0}^{N}\sum_{s=0}^{N} k_{rs} * a_{i-\frac{N-1}{2}+r\, j-\frac{N-1}{2}+s}$$

Para poder realizar el cálculo de los valores de los píxeles que se encuentran al borde de la imagen, se deben rellenar los bordes con píxeles auxiliares de la siguiente forma:

$$A = \begin{pmatrix} a_{\left(-\frac{N-1}{2}\right)\left(-\frac{N-1}{2}\right)} & \cdots & a_{\text{-}1\left(-\frac{N-1}{2}\right)} & a_{0\left(-\frac{N-1}{2}\right)} & \cdots & a_{(W-1)\left(-\frac{N-1}{2}\right)} & a_{W\left(-\frac{N-1}{2}\right)} & \cdots & a_{\left(W+\frac{N-1}{2}\right)\left(-\frac{N-1}{2}\right)} \\ \vdots & \ddots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\ a_{\left(-\frac{N-1}{2}\right)\text{-}1} & \cdots & a_{\text{-}1\text{-}1} & a_{0\text{-}1} & \cdots & a_{(W-1)\text{-}1} & a_{W\text{-}1} & \cdots & a_{\left(W+\frac{N-1}{2}\right)\text{-}1} \\ a_{\left(-\frac{N-1}{2}\right)0} & \cdots & a_{\text{-}10} & a_{00} & \cdots & a_{(W-1)0} & a_{W0} & \cdots & a_{\left(W+\frac{N-1}{2}\right)0} \\ \vdots & \cdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ a_{\left(-\frac{N-1}{2}\right)(H-1)} & \cdots & a_{\text{-}1(H-1)} & a_{0(H-1)} & \cdots & a_{(W-1)(H-1)} & a_{W(H-1)} & \cdots & a_{\left(W+\frac{N-1}{2}\right)(H-1)} \\ a_{\left(-\frac{N-1}{2}\right)H} & \cdots & a_{\text{-}1H} & a_{0H} & \cdots & a_{(W-1)H} & a_{WH} & \cdots & a_{\left(W+\frac{N-1}{2}\right)H} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ a_{\left(-\frac{N-1}{2}\right)\left(H+\frac{N-1}{2}\right)} & \cdots & a_{\text{-}1\left(H+\frac{N-1}{2}\right)} & a_{0\left(H+\frac{N-1}{2}\right)} & \cdots & a_{(W-1)\left(H+\frac{N-1}{2}\right)} & a_{W\left(H+\frac{N-1}{2}\right)} & \cdots & a_{\left(W+\frac{N-1}{2}\right)\left(H+\frac{N-1}{2}\right)} \end{pmatrix}$$

Hay distintas maneras de realizar este rellenado (una forma simple es rellenando con ceros o unos) pero en este caso se rellenarán con los valores de los píxeles adyacentes resultando la siguiente matriz de imagen:

$$A = \begin{pmatrix} a_{00} & \cdots & a_{00} & a_{00} & a_{10} & \cdots & a_{(W-1)0} & a_{(W-1)0} & \cdots & a_{(W-1)0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{00} & \cdots & a_{00} & a_{00} & a_{10} & \cdots & a_{(W-1)0} & a_{(W-1)0} & \cdots & a_{(W-1)0} \\ a_{00} & \cdots & a_{00} & a_{00} & a_{10} & \cdots & a_{(W-1)0} & a_{(W-1)0} & \cdots & a_{(W-1)0} \\ a_{01} & \cdots & a_{01} & a_{01} & a_{11} & \cdots & a_{(W-1)1} & a_{(W-1)1} & \cdots & a_{(W-1)1} \\ \vdots & \cdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ a_{0(H-1)} & \cdots & a_{0(H-1)} & a_{0(H-1)} & a_{1(H-1)} & \cdots & a_{(W-1)(H-1)} & a_{(W-1)(H-1)} & \cdots & a_{(W-1)(H-1)} \\ a_{0(H-1)} & \cdots & a_{0(H-1)} & a_{0(H-1)} & a_{1(H-1)} & \cdots & a_{(W-1)(H-1)} & a_{(W-1)(H-1)} & \cdots & a_{(W-1)(H-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{0(H-1)} & \cdots & a_{0(H-1)} & a_{0(H-1)} & a_{1(H-1)} & \cdots & a_{(W-1)(H-1)} & a_{(W-1)(H-1)} & \cdots & a_{(W-1)(H-1)} \end{pmatrix}$$

Usando el operador de Sobel con kernel de tamaño $3x3$, las derivadas G_x y G_y vienen dadas por las siguientes multiplicaciones de matrices:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A, \; G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

Entonces, el ángulo y la magnitud del gradiente de cada píxel a_{ij} vienen dadas por:

$$\theta(i,j) = atan2\big(G_y(i,j), G_x(i,j)\big)$$

$$|G(i,j)| = \sqrt{G_x(i,j)^2 + G_y(i,j)^2}$$

3.2 Histogramas de Gradientes Orientados (HOG) y descriptores de HOG

El histograma de una imagen divide el rango de valores posibles de los píxeles de la imagen en una serie de sub-rangos o clases – de mismo o distinto tamaño entre ellos – (e.g. Dado el rango de valores de píxel [0, 255], se realiza una división en ocho clases del mismo tamaño: [0, 32), [32, 64), [64, 96)… [224, 255]), y almacena en cada clase la frecuencia de píxeles con un valor comprendido entre ese sub-rango, es decir, el número de píxeles en la imagen cuyo valor está entre los valores de inicio y fin de cada sub-rango.

El Histograma de Gradientes Orientados de una imagen (del inglés *Histogram of Oriented Gradients - HOG*) tiene como rango de valores posibles las distintas orientaciones que pueden tomar los gradientes de los píxeles, i.e. los distintos grados que pueden tomar sus ángulos de gradiente (e.g. [-90°, 90°], [0°, 180°], [0°, 360°]…). Este rango se divide en sub-clases – del mismo tamaño o distintos – (e.g. para el rango [0°, 180°], dividiendo en nueve sub-rangos: [0°, 20°), [20°, 40°)… [160°, 180°]), y se almacena en cada uno de ellas la suma de las magnitudes de gradiente de los píxeles cuyo ángulo de gradiente se encuentra comprendido entre esos valores.

Partiendo del concepto de los HOG, se puede obtener más información de una imagen por medio de un Descriptor de HOG (del inglés HOG Descriptor). En éste, la imagen se divide en un cierto número de sub-imágenes del mismo tamaño, denominadas *celdas*, y éstas se agrupan en *bloques* con un mismo número de celdas de ancho y alto todos ellos. Además, estos bloques se encuentran solapados de forma que el avance de bloques horizontalmente se realiza eliminando la columna de celdas de la izquierda y añadiendo la columna de la derecha y, verticalmente, eliminando la fila de celdas de arriba y añadiendo la fila de celdas de abajo. De este modo, dados una imagen A de tamaño WxH; un tamaño de celda $C_W x C_H$ con $W \ mod \ C_W = 0$ y $H \ mod \ C_H = 0$; y un tamaño de bloque en celdas $B_W x B_H$; el ancho y alto de la imagen en celdas, W_C y H_C, y el número de bloques distribuidos horizontalmente y verticalmente, N_{BW} y N_{BH}, se calculan:

$$W_C = \frac{W}{C_W} \quad , \quad H_C = \frac{H}{C_H}$$

$$N_{BW} = 1 + W_C - B_W \quad , \quad N_{BH} = 1 + H_C - B_H$$

Y por tanto, el número total de celdas N_C y el número total de bloques N_B resultantes de la imagen A será igual a:

$$N_C = W_C * H_C$$

$$N_B = N_{BW} * N_{BH}$$

Y la distribución de celdas y bloques es la siguiente:

$$A = \begin{pmatrix} a_{00} & a_{10} & \cdots & a_{(W-1)0} \\ a_{01} & a_{11} & \cdots & a_{(W-1)1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{0(H-1)} & a_{1(H-1)} & \cdots & a_{(W-1)(H-1)} \end{pmatrix}$$

$$C = \begin{pmatrix} C_{00} & C_{10} & \cdots & C_{(W_C-1)0} \\ C_{01} & C_{11} & \cdots & C_{(W_C-1)1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{0(H_C-1)} & a_{1(H_C-1)} & \cdots & a_{(W_C-1)(H_C-1)} \end{pmatrix}$$

$$B = \begin{pmatrix} B_{00} & B_{10} & \cdots & B_{(N_{BW}-1)0} \\ B_{01} & B_{11} & \cdots & B_{(N_{BW}-1)1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{0(N_{BH}-1)} & B_{1(N_{BH}-1)} & \cdots & B_{(N_{BW}-1)(N_{BH}-1)} \end{pmatrix}$$

A partir de esta estructuración de la imagen, el descriptor de HOG calcula de forma independiente el HOG de cada celda y cada bloque agrupa los HOGs de sus celdas correspondientes. Entonces, el número de HOGs que contiene un descriptor será:

$$N_{HOG} = B_W * B_H * N_B$$

Y si se divide el HOG en n clases, dado que cada bloque contiene $B_W * B_H$ HOGs, entonces el número total de valores N_V que se tomará de la imagen A será:

$$N_V = n * N_{HOG}$$

Para el cálculo del modelo de detección, sobre la colección de imágenes se debe calcular el descriptor de HOG de cada imagen, etiquetando cada descriptor como positivo si es una imagen de mano (*"+1"*) o negativo si no lo es (*"-1"*).

El cálculo de los descriptores de HOG puede tener un coste en tiempo de computación bastante grande pues requiere del cálculo de un HOG por cada celda. Para agilizar esto se puede usar una técnica denominada Integral de HOG por el cual se mejora el coste de

computación a cambio de aumentar el coste en memoria. Para calcular la Integral de HOG se necesitan primero n matrices auxiliares de tamaño igual a la imagen de entrada A, WxH, y donde n es el número de clases en el HOG. Cada una de estas matrices estará asociada a una única clase, es decir, a un único rango de ángulos de gradiente, y donde cada uno de sus píxeles cumple:

$$M(b,i,j) = \begin{cases} |G(i,j)|, & \theta(i,j) \in R(b) \\ 0, & eoc \end{cases}$$

donde $b \in \{0,1,\dots,n-1\}$ indica la clase, $i \in \{0,1,\dots,W-1\}$ indica la columna de la matriz, $j \in \{0,1,\dots,H-1\}$ indica la fila de la matriz y $R(b)$ es el rango de valores de ángulo asociado a la clase b. En la ilustración 3.1 se puede ver una representación gráfica de estas matrices.

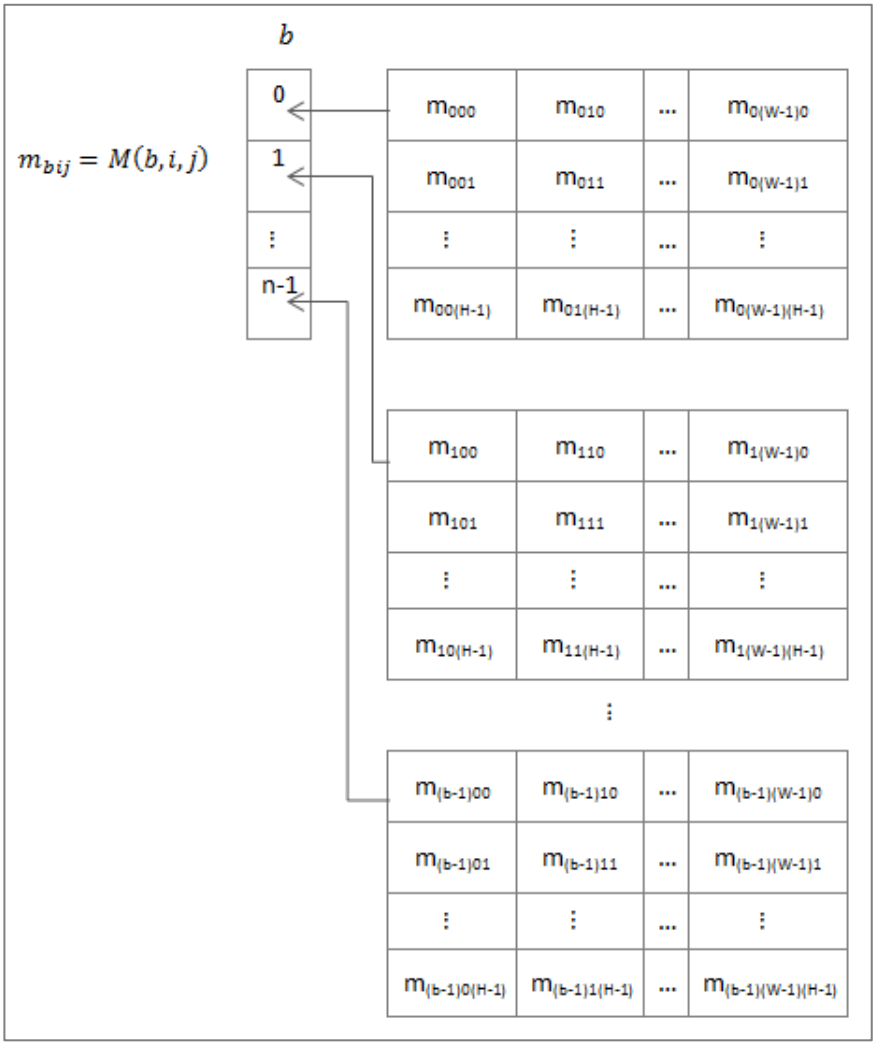

Ilustración 3.1 - Matrices auxiliares para el cálculo de una Integral de HOG

La Integral de HOG (véase ilustración 3.2) está compuesta de n matrices de tamaño $(W+1)x(H+1)$, cada una de ellas asociadas a una clase y donde cada píxel (i,j) indica la suma de las magnitudes del rectángulo de la imagen A con esquina superior izquierda $(0,0)$ y tamaño ixj. Éstas matrices se calculan a partir de las matrices M anteriores y se puede definir una Integral de HOG como la siguiente función recursiva:

$$IH(b,i,j) = \begin{cases} 0, & i = 0 \vee j = 0 \\ IH(b,i,j-1) + \sum_{k=0}^{i} B(b,k,j), & eoc \end{cases}$$

De esta forma, para el cálculo del HOG de cada celda de un Descriptor de HOG se deberá realizar una simple resta por cada clase por medio de la siguiente ecuación:

$$C(b,i,j,W_C,H_C) = IH(b,i+W_C,j+H_C) - IH(b,i+W_C,j) - IH(b,i,j+H_C) + IH(b,i,j)$$

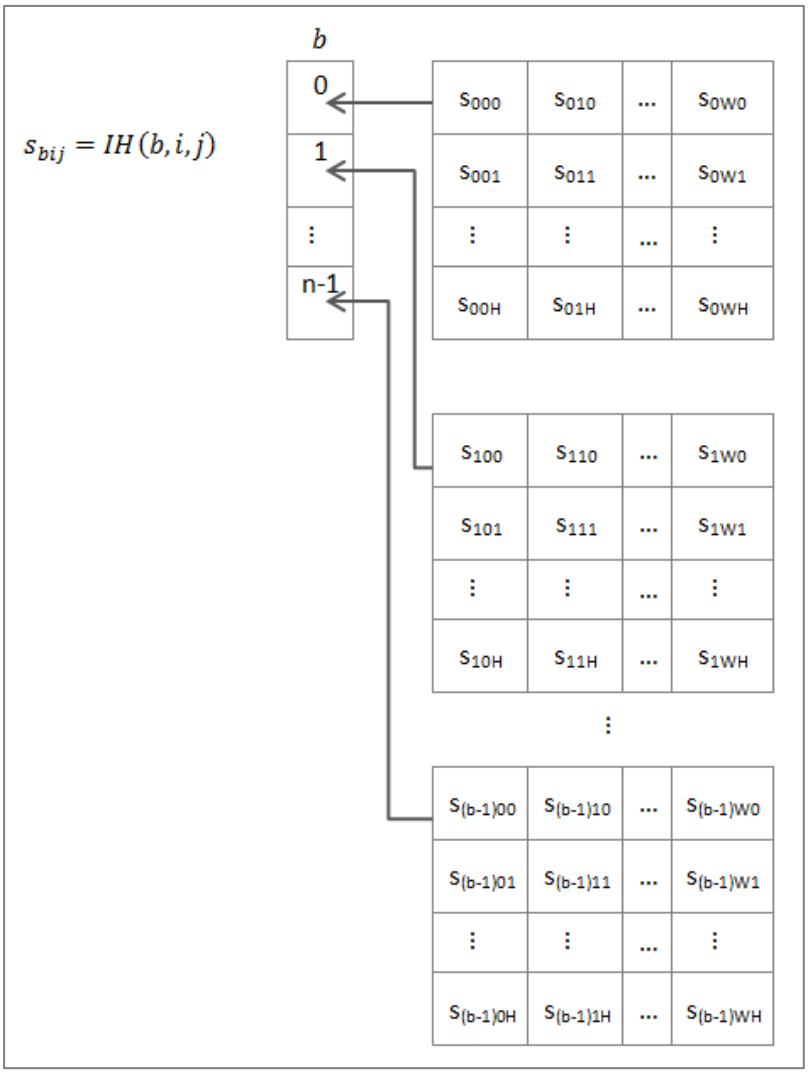

Ilustración 3.2 - Integral de HOG

3.3 Máquina de vectores de soporte (SVM)

Una máquina de vectores de soporte [4] (del inglés *Support Vector Machine - SVM*) es un método de aprendizaje supervisado que, a partir de unas muestras de entrada, reconoce patrones y permite resolver problemas de clasificación y de regresión. En este caso, se usará este algoritmo para poder identificar manos, el cual se trata de un problema de clasificación de dos clases: una clase asociada a las manos y otra a todo aquello que no lo sea.

Se puede concebir de una forma abstracta el problema de clasificación de dos clases como se aprecia en la ilustración 3.3. Partiendo de un conjunto de muestras iniciales positivas y negativas con n características cada uno, se posiciona cada muestra como un punto en un espacio de n dimensiones. El objetivo es encontrar, si es posible, una línea o hiperplano que separe los puntos de ambas clases en dos grupos, y haciendo que la distancia global entre los puntos de las mismas sea lo máximo posible.

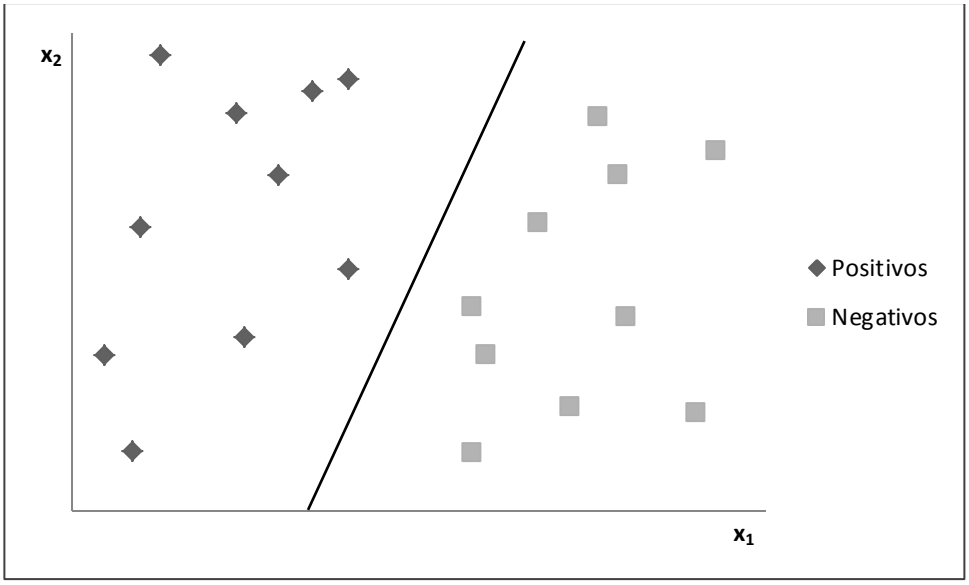

Ilustración 3.3 - Hiperplano de separación de dos clases

Ahora se procederá a profundizar un poco más en las SVM realizando una descripción matemática del funcionamiento de una máquina de soporte para este caso particular de clasificación en dos clases.

Dada una entrada compuesta por M muestras de entrenamiento $x_i \in \mathbb{R}^n$ $(i = 1, \dots, M)$ pertenecientes a las clases C_1 o C_2 y las etiquetas asociadas $y_i \in \{1, -1\}$, siendo $y_i = 1$ si x_i pertenece a la clase C_1 y $y_i = -1$ si pertenece a la clase C_2, se dice que las M muestras son *linealmente separables* si podemos determinar su *función de decisión* como:

$$D(x) = w \cdot x + b \tag{3.1}$$

donde $w \in \mathbb{R}^n$, $b \in \mathbb{R}$ y se cumple que para $i = 1, \dots, M$

$$w \cdot x_i + b \begin{cases} > 0 & para\ y_i = 1 \\ < 0 & para\ y_i = -1 \end{cases} \tag{3.2}$$

Suponiendo que los datos de entrada son linealmente separables, ninguna entrada x_i satisface $w \cdot x_i + b = 0$. Para controlar la separabilidad entre las clases y poder separar las muestras en función del valor de su etiqueta, se sustituyen las anteriores desigualdades por las siguientes:

$$w \cdot x_i + b \begin{cases} \geq 1 & para\ y_i = 1 \\ \leq -1 & para\ y_i = -1 \end{cases} \tag{3.3}$$

Simplificando las ecuaciones de desigualdad de (3.3):

$$y_i(w \cdot x_i + b) \geq 1 \quad para\ i = 1, \dots, M \tag{3.4}$$

El hiperplano

$$D(x) = w \cdot x + b = c \quad para\ -1 < c < 1 \tag{3.5}$$

crea un *hiperplano de separación* entre los datos de entrada x_i $(i = 1, \dots, M)$. Cuando $c = 0$, el hiperplano de separación se encuentra en el medio de los hiperplanos con $c = 1$ y $c = -1$. Se denomina *margen* a la distancia existente entre el hiperplano de separación y la muestra de entrenamiento más cercana al mismo. Suponiendo que los hiperplanos $D(x) = 1$ y $D(x) = -1$ contienen al menos una muestra cada una, $D(x) = 0$ tiene el máximo margen para $-1 < c < 1$. El hiperplano con máximo margen se denomina *óptimo hiperplano de separación*.

El vector w es ortogonal al hiperplano de separación y, por tanto, la línea que forman una muestra de entrada x y el vector w es también ortogonal a la misma. Esta línea viene definida por la siguiente ecuación: $\frac{d}{\|w\|} w + x$ donde $|d|$ es la distancia entre x y el

hiperplano. Esta línea cortará el hiperplano de separación cuando se satisfaga

$$D\left(\frac{d}{\|w\|}w + x\right) = 0 \tag{3.6}$$

Despejando esta ecuación se obtiene el valor de la distancia d:

$$D\left(\frac{d}{\|w\|}w + x\right) = w^T \cdot \left(\frac{d}{\|w\|}w + x\right) + b = \frac{d}{\|w\|}w^T \cdot w + w^T \cdot x + b$$

$$= \frac{d}{\|w\|}\|w\|^2 + w^T \cdot x + b = d\|w\| + w^T \cdot x + b = d\|w\| + D(x) = 0$$

$$\Rightarrow \quad d = -\frac{D(x)}{\|w\|}$$

Por tanto, para los hiperplanos $D(x) = 1$ y $D(x) = -1$, su distancia al hiperplano de separación es $\frac{1}{\|w\|}$ en ambos casos por lo que la distancia entre ambos hiperplanos es $\frac{2}{\|w\|}$ (Véase la ilustración 3.4).

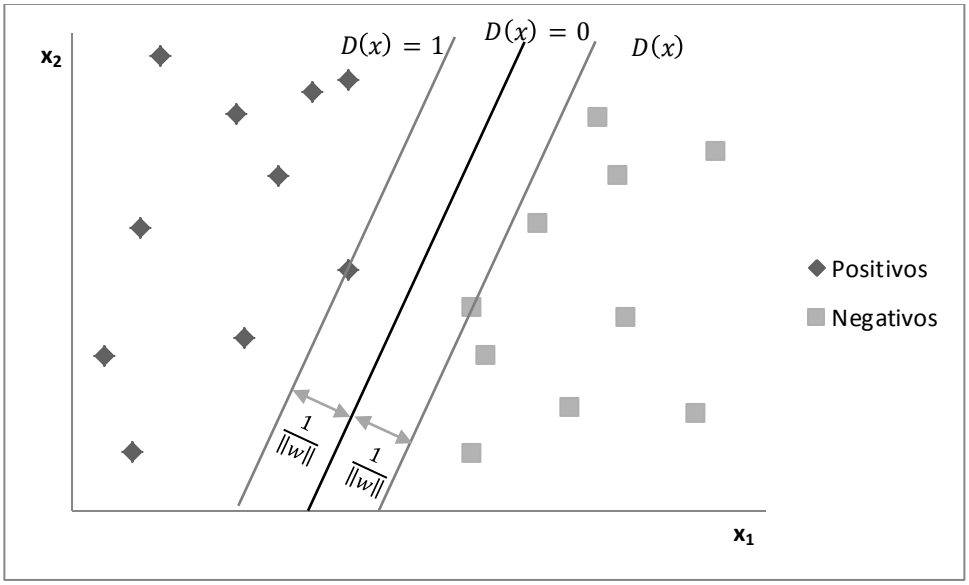

Ilustración 3.4 - Hiperplano de separación óptimo y sus distancias de margen

El objetivo será maximizar esta distancia por lo que se está ante el siguiente problema de optimización:

$$maximizar \quad f(w,b) = \frac{2}{\|w\|}$$
$$tal\ que\ y_i(w \cdot x + b) \geq 1 \quad para \quad i = 1, \ldots, M \tag{3.7}$$

Un problema de *programación cuadrática* (del inglés *quadratic programming*-QP) es un tipo especial de problema de optimización matemática en el cual se dispone de una función objetivo de orden cuadrático de múltiples variables que se quiere optimizar (maximizar o minimizar) y que se encuentra sujeto a una serie de restricciones sobre esas variables de orden lineal. La forma general de un QP-problema de minimización es el siguiente:

$$minimizar\ g(x) = \frac{1}{2}x^T Qx + c^T x$$
$$tal\ que\ \ Ax \leq b\ con\ x \geq 0 \tag{3.8}$$

donde x es el vector n-dimensional de las n variables a calcular, c es un vector n-dimensional que tiene por valor los coeficientes de los términos lineales de la función objetivo, Q es una matriz simétrica nxn que incluye los coeficientes de los términos cuadráticos de la función y las restricciones son definidas por una matriz mxn A de coeficientes aplicadas al vector de entrada y un vector b m-dimensional de valores para la desigualdad. Se supone que existe una solución que satisface todas las restricciones.

Entonces, se puede convertir el problema de optimización (3.7) en el siguiente QP-problema sobre w y b:

$$minimizar \quad g(w,b) = \frac{1}{2}\|w\|^2 \tag{3.9}$$
$$tal\ que\ y_i(w \cdot x + b) \geq 1 \quad para \quad i = 1, \dots, M \tag{3.10}$$

Dado que se ha supuesto que los datos de entrada son linealmente separables, se sabe que existen w y b que satisfacen (3.10). A las soluciones que satisfacen (3.10) se las denomina *soluciones factibles*. La solución a las restricciones puede no ser única pero el valor de la solución objetivo si que lo es.

El método de los *multiplicadores de Lagrange* permite convertir un problema de optimización restringido de n variables a otro no restringido de $n + m$ variables donde m es el número de restricciones. Cada una de estas variables asociadas a las restricciones se las denomina *multiplicador de Lagrange*.

Para que el problema sea más fácil de manejar, se convierte la expresión restringida (3.9) y (3.10) por una expresión sin restricciones usando multiplicadores de Lagrange:

$$h(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{M} \alpha_i \{y_i(w \cdot x_i + b) - 1\}$$
$$= \frac{1}{2}\|w\|^2 - \sum_{i=1}^{M} \alpha_i y_i (w \cdot x_i + b) + \sum_{i=1}^{M} \alpha_i \quad (3.11)$$

donde $\alpha = (\alpha_1, \dots, \alpha_M)$ y α_i es un multiplicador de Lagrange no negativo.

La solución óptima de (3.10) se da cuando es minimizada con respecto a w, maximizada con respecto a α_i y maximizada o minimizada con respecto a b dependiendo del signo de $\sum_{i=1}^{M} \alpha_i y_i$.

El método de *Karush-Kuhn-Tucker* (*KKT*) es utilizado para derivar condiciones que comprueban si una solución dada es óptima o no. Aplicándolo a este problema se obtienen las siguientes condiciones de (KKT):

$$\frac{\partial h(w,b,\alpha)}{\partial w} = 0 \quad (3.12)$$

$$\frac{\partial h(w,b,\alpha)}{\partial b} = 0 \quad (3.13)$$

$$\alpha_i\{y_i(w \cdot x_i + b) - 1\} = 0 \quad para \;\; i = 1, \dots, M \quad (3.14)$$

$$\alpha_i \geq 0 \quad para \;\; i = 1, \dots, M \quad (3.15)$$

Por (3.13) y (3.14) se sabe que, o bien $\alpha_i = 0$, o bien $\alpha_i \neq 0$ y $y_i(w \cdot x + b) = 1$. Aquellas muestras de entrada x_i con $\alpha_i \neq 0$ son los denominados *vectores de soporte*.

A partir de (3.10), se deduce con (3.11) y (3.12) respectivamente que

$$\frac{\partial h(w,b,\alpha)}{\partial w} = w - \sum_{i=1}^{M} \alpha_i y_i x_i = 0 \;\Rightarrow\; w = \sum_{i=1}^{M} \alpha_i y_i x_i \quad (3.16)$$

y

$$\frac{\partial h(w,b,\alpha)}{\partial b} = \sum_{i=1}^{M} \alpha_i y_i = 0 \quad (3.17)$$

Sustituyendo (3.15) y (3.16) en (3.10) se obtiene

$$h(w,b,\alpha) = \frac{1}{2}\sum_{i=1}^{M}\alpha_i y_i\, x_i \sum_{j=1}^{M}\alpha_j y_j\, x_j - \sum_{i=1}^{M}\alpha_i y_i \left(\sum_{j=1}^{M}\alpha_j y_j\, x_j \cdot x_i + b\right) + \sum_{i=1}^{M}\alpha_i$$

$$= \frac{1}{2}\sum_{i,j=1}^{M}\alpha_i\alpha_j y_i y_j x_i x_j - \sum_{i,j=1}^{M}\alpha_i\alpha_j y_i y_j x_i x_j - \sum_{j=1}^{M}\alpha_j y_j\, b \qquad (3.18)$$

$$+ \sum_{i=1}^{M}\alpha_i = -\frac{1}{2}\sum_{i,j=1}^{M}\alpha_i\alpha_j y_i y_j x_i x_j - 0 + \sum_{i=1}^{M}\alpha_i$$

$$\Rightarrow h(\alpha) = \sum_{i=1}^{M}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{M}\alpha_i\alpha_j y_i y_j x_i x_j$$

Se está por tanto ante el siguiente problema de optimización dual:

$$maximizar \;\; h(\alpha) = \sum_{i=1}^{M}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{M}\alpha_i\alpha_j y_i y_j x_i x_j \qquad (3.19)$$

$$tal\ que \sum_{i=1}^{M}\alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad para \;\; i = 1, \ldots, M \qquad (3.20)$$

Resolviendo este problema (mediante programación dinámica por ejemplo) se obtiene una solución para los multiplicadores de Lagrange $\alpha = (\alpha_1, \ldots, \alpha_M)$ y todas aquellas muestras x_i asociadas a un α_i positivo serán los vectores de soporte. Entonces, a partir de (3.15) la función de decisión viene dada por

$$D(x) = \sum_{i \in S}\alpha_i y_i x_i \cdot x + b \qquad (3.21)$$

donde S es el conjunto de índices de las muestras que son vectores de soporte, y de la condición KKT (3.13) se deduce

$$y_i(w \cdot x_i + b) = 1$$

$$b = \frac{1}{y_i} - w \cdot x_i = y_i - w \cdot x_i \qquad (3.22)$$

Para una mayor precisión de los cálculos, es mejor tomar como valor de b una media aritmética sobre todo los vectores de soporte:

$$b = \frac{1}{|S|} \sum_{i \in S} y_i - w \cdot x_i \tag{3.23}$$

Y finalmente se tiene ya una función de decisión que para una muestra indefinida x realizará la clasificación del siguiente modo:

$$\begin{cases} Clase\ C_1 & si\ \ D(x) > 0 \\ Clase\ C_2 & si\ \ D(x) < 0 \end{cases} \tag{3.24}$$

Si se cumple $D(x) = 0$, entonces la muestra x se encuentra en la frontera y es inclasificable.

El caso explicado es el denominado *Máquina de vectores de soporte de margen duro* (del inglés *Hard-Margin*) en el cual los datos de entrada son separados linealmente sin problemas. Sin embargo, cuando no se pueden separar linealmente, este método no da soluciones factibles. Añadiendo una cierta compensación entre la maximización del margen y la minimización del error de clasificación se puede solucionar este problema. Se está entonces ante una *Máquina de vectores de soporte de margen blando* y a esta compensación se la denomina margen *C*.

Se puede mejorar el tiempo de computación que requieren las *Máquina de vectores de soporte* para su entrenamiento realizando el cálculo con ayuda de kernels. Estos kernels pueden ser de múltiples tipos y cada uno de ellos ofrece resultados distintos: lineales, polinomiales, funciones de base radial, redes neuronales de tres niveles, etc

3.4 Proceso de aprendizaje

Una vez que se ha obtenido el primer modelo de detección se debe testear con un conjunto de imágenes de prueba para comprobar su eficiencia. Si una imagen positiva no logró ser detectada (falso negativo), ésta deberá ser añadida como imagen positiva a la colección de imágenes de muestra a partir de la cual se calculó el detector. Si una imagen negativa fue detectada (falso negativo), ésta deberá ser añadida a la colección como imagen negativa. Cuando una imagen positiva es detectada o una negativa no es detectada entonces el funcionamiento será correcto y no se deberá hacer nada con esas imágenes. Tras realizar esto con todas las imágenes de testeo, se volverá a usar el SVM con las nuevas imágenes

añadidas para calcular un nuevo modelo y se repetirá el test. Se repetirá todo este proceso hasta que se obtenga una precisión del detector razonable (superior al 95%).

3.5 Auto-detección de manos

Una vez se dispone de la función de decisión para la clasificación, definida por su vector w y el escalar b, se puede proceder a la detección o localización de manos. Este algoritmo recibirá a la entrada una imagen I, un clasificador de manos C y un factor de escalas S_r y devolverá un conjunto con los rectángulos que incluyen a las detecciones.

Suponiendo que el tamaño de las muestras usadas es $W_w x H_w$ y el tamaño de la imagen de entrada a detectar es $W_I x H_I$ siendo $W_w \leq W_I$ y $H_w \leq H_I$, el algoritmo es el siguiente:

1º. Comenzando por una escala $S_0 = 1$, se calculan todas las distintas escalas posibles multiplicando por el factor de escala S_r de forma que $S_i = S_{i-1} * S_r$. Se parará cuando $S_i * W_w > W_I$ o $S_i * H_w > H_I$. Sea $S_e = min\left(\frac{W_I}{W_w}, \frac{H_I}{H_w}\right)$, el número de escalas posibles n es igual a

$$n = floor\left(\log_{S_r} S_e\right) + 1 = floor\left(\frac{\log S_e}{\log S_r}\right) + 1$$

2º. Sea $S = [S_0, S_1, \ldots, S_{n-1}]$ y R un conjunto de rectángulos que crece incluyendo los rectángulos de las detecciones positivas. Para cada escala S_i se realizan los siguientes pasos:

- Se re-escala la imagen de entrada.
- Se recorre la imagen re-escalada con un tamaño de ventana igual a la muestra $W_w x H_w$ y se calcula para cada uno su vector de características HOG. Aplicando el clasificador C se comprueba si pertenece o no a la clase manos. Si pertenece se añade la posición y el tamaño del rectángulo de la detección re-escalada a S_i; i.e. si la detección se realizó en el rectángulo de esquina superior izquierda (x, y) y tamaño $W_w x H_w$, en R añadiremos el rectángulo de esquina superior izquierda $(x * S_i, y * S_i)$ y tamaño $(W_w * S_i) x (H_w * S_i)$.

3º. Finalmente, se comprueba en el conjunto R qué rectángulos tienen posiciones y tamaños parecidos y se combinan.

3.6 Resultados de tiempo

Las pruebas de detección se realizaron sobre un computador con las siguientes características:

- **Procesador:** Intel® Core™ i5 430M (2.27 GHz, 3 MB L3 cache)
- **RAM:** 4,00 GB DDR3
- **GPU:** ATI Mobility Radeon™ HD 5650 (Hasta 2736 MB HyperMemory™)

El tamaño de las imágenes de muestra usados para entrenar el detector es de 64x64 píxeles, un tamaño de celda de 8x8 píxeles, un tamaño de bloque de 2x2 celdas, un rango de valores [0°, 180°] y un HOG dividido en 9 clases, por lo que el número de valores que se toma de cada imagen se calcula del siguiente modo:

$$W = 64\,px, H = 64\,px; \quad C_W = 8\,px, C_H = 8\,px; \quad B_W = 2\,celdas, B_H = 2\,celdas; \quad n = 9$$

$$W_C = \frac{W}{C_W} = \frac{64}{8} = 8\,celdas, \qquad H_C = \frac{H}{C_H} = \frac{64}{8} = 8celdas$$

$$N_{BW} = 1 + W_C - B_W = 1 + 8 - 2 = 7\ bloques$$

$$N_{BH} = 1 + H_C - B_H = 1 + 8 - 2 = 7\ bloques$$

$$N_C = W_C * H_C = 8 * 8 = 64\ celdas$$

$$N_B = N_{BW} * N_{BH} = 7 * 7 = 49\ bloques$$

$$N_{HOG} = B_W * B_H * N_B = 2 * 2 * 49 = 196\ HOGs$$

$$N_V = n * N_{HOG} = 9 * 196 = 1764\ valores$$

Tras varios entrenamientos y testeos del detector, el número final de imágenes que se usaron fue de 1.987 imágenes positivas y 7.431 imágenes negativas por lo que para realizar el aprendizaje se tomaron 9.478 descriptores de HOG de 1.764 valores cada una (un total de 16.613.352 valores). El tiempo tomado para calcular estos 9.478 descriptores fue de 3 minutos y 35 segundos.

Usando el programa SVM^{light} para realizar el aprendizaje, el tiempo requerido para el entrenamiento del último detector fue de 3 minutos y 13 segundos, donde se ajustaron los parámetros a los siguientes valores: C = 0.01, kernel = lineal. A partir de la base de imágenes de testeo, el nivel de precisión marcado para el detector es del 97,94%.

El tiempo de detección usando este detector aplicado a una imagen de 640x480 píxeles varía entre 500 ms y 600 ms.

Se realizaron pruebas de rendimiento de la fase de auto-detección en otras computadoras con características técnicas distintas para conocer hasta que punto la aplicación podía ser usada en tiempo real, la cual era una de las premisas sobre las que se asienta la idea de este proyecto.

Sobre una computadora portátil con las siguientes características:

- **Procesador:** Intel® Atom™ Processor N450 (1,66 GHz, 512KB cache)
- **RAM:** 1GB
- **GPU:** Intel Graphics Media Accelerator 3150

se obtienen unos tiempos de ejecución que oscilan entre 2 y 3 segundos, en el caso de realizar la detección sobre una imagen de 640x480 píxeles. Estos tiempos son comprensibles dada las características técnicas peores de la máquina en cuestión.

A pesar de que los tiempos que se obtienen unos son altos, no se considera que se pierda la interacción en tiempo real, ya que esta parte del algoritmo sólo se ejecuta mientras que no sea capaz de localizar una mano en toda la imagen. En el momento en el que se localiza la posición de la mano y se empieza a seguir, la etapa de detección no se ejecuta, consiguiendo unos tiempos de ejecución que permiten la interacción en tiempo real.

3.7 Dificultades encontradas

En un principio se optó por tomar muestras de tamaño 128x128 píxeles puesto que a mayores muestras, más precisas son las descripciones de los objetos a detectar en los descriptores de HOG (la precisión del detector era del 98,92%). Este tamaño producía unos retardos de detección de entre 800 ms y 900 ms, los cuales no eran viables. Por ello se redujo el tamaño de muestra obteniendo claras mejoras de tiempo y con una perdida de precisión razonable (~1%).

Sin embargo, un retardo de 500 ms tampoco es un valor válido ya que ello implica una tasa de frames de sólo 2 fps. Para mejorar esto, se aprovechó una característica del objetivo de este proyecto: el seguimiento. Dado que lo que se desea realizar es reconocer las manos y sus movimientos, en cuanto se detecta una mano en el siguiente frame la mano se encontrará en una posición cercana a la actual. Entonces, la búsqueda de la mano puede restringirse a

una zona próxima a la de la detección actual con lo que la detección será más rápida por ser el área de búsqueda menor. Con este método se reduce bastante el tiempo de retardo pero, ahora, éste es dependiente de la distancia de la mano a la webcam. Cuanto más cerca esté de ella, mayor es el área de búsqueda y, por tanto, mayor también el tiempo de retardo; y viceversa. Sin embargo, para el caso de uso habitual la distancia suele ser siempre parecida y el retardo se encuentra en torno a los 100 ms lo cual mejora la tasa de frames a 10 fps.

4. Obtención de características

El objetivo de esta etapa es diferenciar los píxeles de la imagen que pertenecen al contorno del objeto de estudio (en nuestro caso la mano) del resto de píxeles de la imagen. Posteriormente, a partir de él se obtiene una estructura que describen puntos y características relevantes de la mano.

Se realizan por tanto tres fases: adaptación de la imagen, obtención del contorno y cálculo de puntos estructurales.

Es necesario antes de calcular el contorno una fase previa de *adaptación de la imagen* de entrada:

- La imagen de la cámara codificada en tres canales RGB, es convertida en una imagen de un solo canal de escala de grises para mejorar el rendimiento y simplificar los cálculos sucesivos.
- Sobre esta imagen se realiza una ecualización del histograma con lo cual se maximiza el contraste. Esta técnica aumenta la probabilidad de encontrar el contorno correctamente.

Tras estos pasos previos se comienza con la fase de *obtención del contorno*. Esta fase recibe como entrada la imagen obtenida por la cámara y una región, calculada en la etapa anterior, donde se ha detectado la presencia de una mano. Con estos datos se obtiene el contorno de la mano que permitirá la localización de sus puntos estructurales y el cálculo de sus características descriptivas.

Con el fin de detectar los bordes se utilizan los gradientes de intensidad. De este modo los puntos que tengan una mayor variación de intensidad respecto a sus vecinos coincidirán, a excepción de brillos y sombras, con el contorno.

Debido a que únicamente nos interesan bordes de la mano y en la imagen aparecerán también bordes del brazo, acotamos dichos bordes artificialmente en la zona inferior de la región que se ha recibido como entrada.

La imagen se binariza en función de que el valor del píxel se encuentre por encima de un mínimo valor de variación (umbral). A los píxeles que cumplen la condición anterior se les da un valor de blanco (255) y negro al resto (0).

Sobre la imagen binarizada encontramos los puntos pertenecientes al contorno. Estos puntos han de cumplir las condiciones de continuidad espacial. En el caso de encontrar varios contornos no conexos se tomarán los contornos más externos, no contenidos en otros, y que tengan un perímetro que se encuentre dentro de un rango de valores posibles.

En la fase de *cálculo de puntos estructurales* se efectúa un test para comprobar que los contornos pertenecen, efectivamente, a una mano. Para realizar dicho test se obtienen los puntos que se corresponden con las puntas de los dedos y los espacios interdigitales y se comprueban que cumplan unas determinadas condiciones. Denominaremos a estos puntos como *puntos estructurales*.

Para el cálculo de estos puntos estructurales se realizan los siguientes pasos:

- Se calcula un recubrimiento poligonal convexo sobre el contorno hallado. El resultado será un nuevo contorno en el cual no aparecerán las concavidades producidas por los espacios entre dedos y posibles errores en el cálculo del contorno en la etapa anterior.
- Se calculan los defectos de convexidad, es decir, los puntos pertenecientes al contorno inicial que difieren de los del recubrimiento. Esta serie de puntos tienen un punto inicial, primer punto en el que ambos contornos difieren, un punto final, último punto de la serie antes de que los contornos vuelvan a converger, y un punto de máxima profundidad, punto del contorno inicial más lejano del recubrimiento. Estos tres puntos definen cada defecto de convexidad.

Se obtienen los puntos de los extremos de los dedos y de los espacios interdigitales a partir de los defectos de convexidad.

Obtenidos dichos puntos comprobamos que cumplan unas determinadas condiciones para comprobar que pertenecen a puntos representativos de una mano.

4.1. Adaptación de la imagen

La imagen a color de tres canales con codificación RGB capturada por la cámara es transforma en una imagen en escala de grises de un solo canal. Para conseguirlo se calcula la media ponderada de los tres canales BGR y el resultado es el valor del canal en escala de grises. Es decir, para cada píxel se aplica:

$$RGB[A]\ to\ Gray: Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

El beneficio de realizar esta conversión consiste en reducir el número de píxeles que se tienen que computar, ya que se ha pasado de una imagen con tres canales a una imagen de un solo canal. Esta simplificación se puede realizar gracias al hecho de que, para realizar el cálculo de bordes, no son relevantes los valores concretos de intensidad de cada canal de color primario (Red, Green, Blue) si no más bien la variación de intensidad global que se produce.

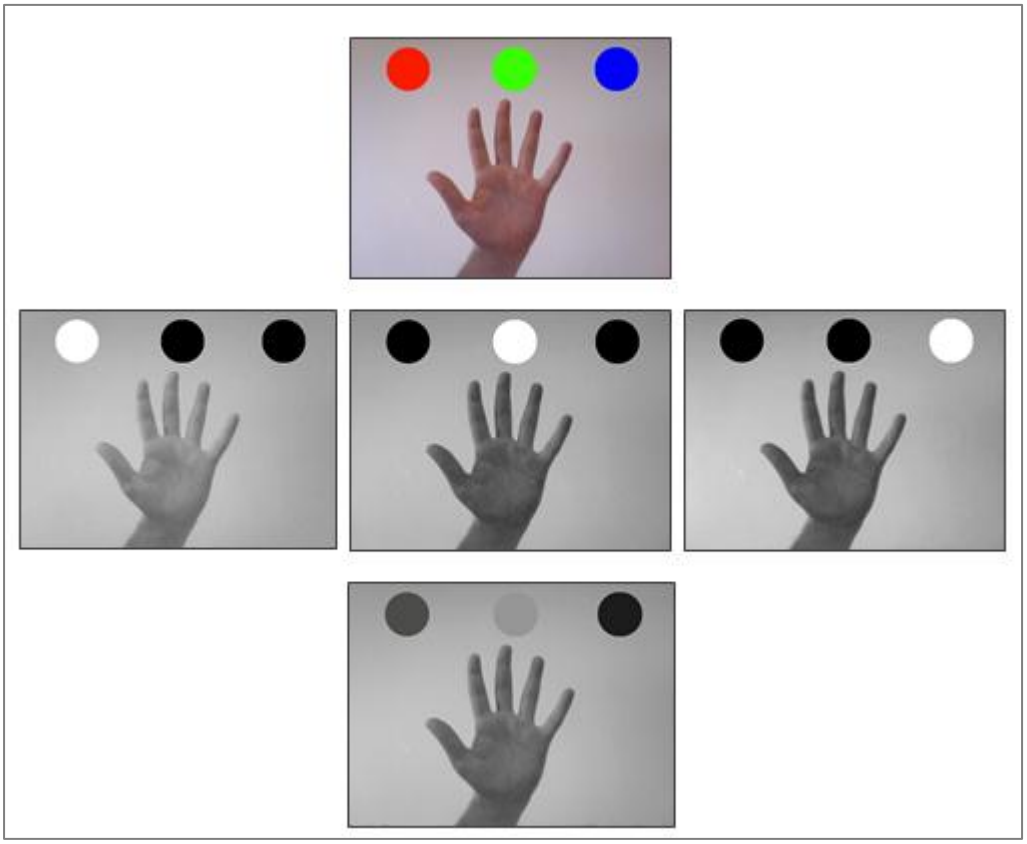

Ilustración 4.1 - Conversión de una imagen RGB a escala de grises

Esta conversión se realiza con la función `cvtColor(img,gray_img,cvt_type)` siendo:

- **img**: parámetro de entrada. Imagen inicial en codificación BGR de tres canales.
- **gray_img:** parámetro de salida. Imagen destino de un solo canal en escala de grises.
- **cvt_type:** parámetro de entrada. Opción del tipo de conversión. En este caso CV_BGR2GRAY, de BGR a escala de grises.

Posteriormente se realiza la *ecualización del histograma* que consiste en, una vez calculado el histograma de la imagen monocroma en escala de grises, aumentar las frecuencias de los valores de intensidad cercanos a valores de intensidad con altas frecuencias y normalizarlo. De este modo se produce un efecto de "expansión del

histograma" con la consecuencia de aumentar el contraste de la imagen.

Para calcular el histograma ecualizado de una imagen se emplea la función de distribución acumulada:

$$H'(i) = \sum_{0 \le j < i} H(j)$$

donde $H(j)$ es el valor de frecuencia de la intensidad j en el histograma sin ecualizar y $H'(i)$ es el valor de distribución acumulada para la intensidad i.

Dado que con esta fórmula se obtendrán valores por encima de 255, se normaliza de modo que el máximo se ajuste a este valor.

Una vez obtenidos los valores de esta distribución, para el valor de intensidad de cada píxel en la posición (x,y) de la imagen se calcula su nuevo valor de intensidad usando su función de distribución acumulada:

$$I_{eq}(x,y) = H'\big(I(x,y)\big)$$

donde $I_{eq}(x,y)$ es el valor de intensidad de la imagen ecualizada en el píxel (x,y), $I(x,y)$ es el valor de intensidad de la imagen original en el píxel (x,y) y H' es la función de distribución acumulada.

Estos pasos se realizan con la función `equalizeHist(gray_img,equal_img)` siendo:

- **gray_img:** parámetro de entrada. Imagen origen en escala de grises.
- **equal_img:** parámetro de salida. Imagen destino ecualizada en escala de grises.

Con esto se consigue aumentar el contraste en la imagen. En el caso de capturar una imagen muy oscura o muy brillante nos permite poder identificar mejor las formas que hay en ellas. El beneficio se obtiene al maximizar la diferencia entre el objeto en primer plano (la mano) y el fondo uniforme, facilitando el posterior cálculo del contorno.

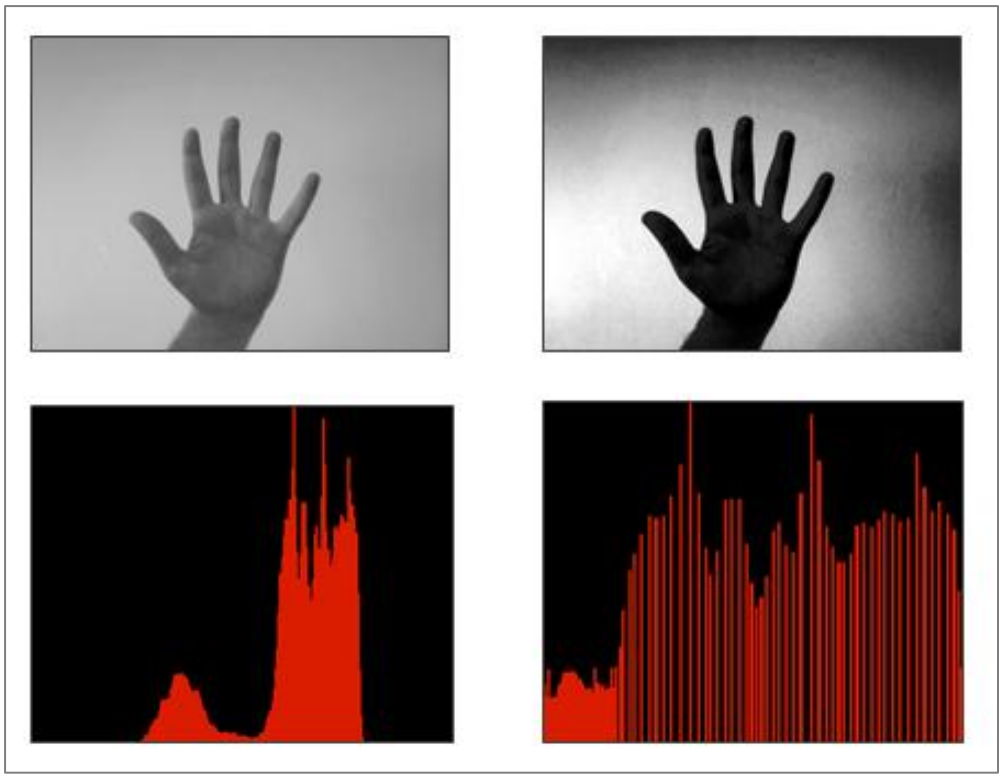

Ilustración 4.2 - Ecualización del histograma de una imagen e histograma correspondiente (a la izquierda sin ecualizar y a la derecha ecualizada)

4.2. Obtención del contorno

El *cálculo de los gradientes de intensidad* nos permite identificar los puntos de la imagen donde hay una mayor variación de intensidad. Por lo general esta variación se maximiza localmente en los píxeles dónde aparece la transición entre la superficie de dos objetos. Asumiendo que el fondo de la imagen sea uniforme y en primer plano aparezca la mano (de un color también uniforme), estos puntos coincidirán con puntos de borde o frontera del objeto de interés.

En la ilustración 4.3 se puede observar la variación de intensidad alrededor de un punto de borde (primera imagen). Si observamos como varía la intensidad en los puntos de la horizontal que pasa sobre él, encontramos el punto de máxima variación (segunda imagen). Representando la derivada de la intensidad, éste será el punto máximo (tercera imagen).

El gradiente se calcula en la dirección horizontal y vertical gracias a una convolución de la imagen con los operadores de Sobel.,

Para tener en cuenta las variaciones en ambas direcciones se suman sus valores en valor absoluto. Para cada píxel se aplica la fórmula:

$$G = |G_x| + |G_y|$$

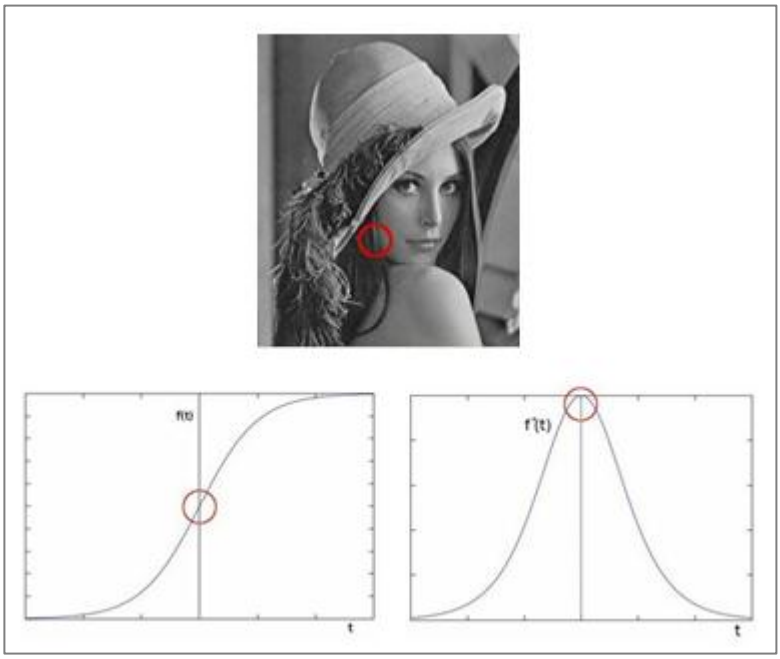

Ilustración 4.3 – Derivada y Gradiente de un píxel

Después de haber obtenido las dos imágenes correspondientes al gradiente horizontal y al gradiente vertical (Ilustración 4.2), se recorren píxel a píxel generando otra imagen que contendrá las magnitudes de variación sumando los valores absolutos de las dos y normalizando para que los valores se encuentren dentro del rango de representación de grises (0-255).

Para el cálculo de los gradientes en ambas direcciones se usa la función `Sobel(gray_img,sobel_img,sobel_depth,dx,dy)` siendo:

- **gray_img:** imagen en escala de grises origen.
- **sobel_img:** imagen en escala de grises destino.
- **sobel_depth:** profundidad de píxel de la imagen destino.
- **dx:** orden de la derivada horizontal. En nuestro caso 1 para el gradiente horizontal y 0 para el vertical.
- **dy:** orden de la derivada vertical. En nuestro caso 0 para el gradiente horizontal y 1 para el vertical.

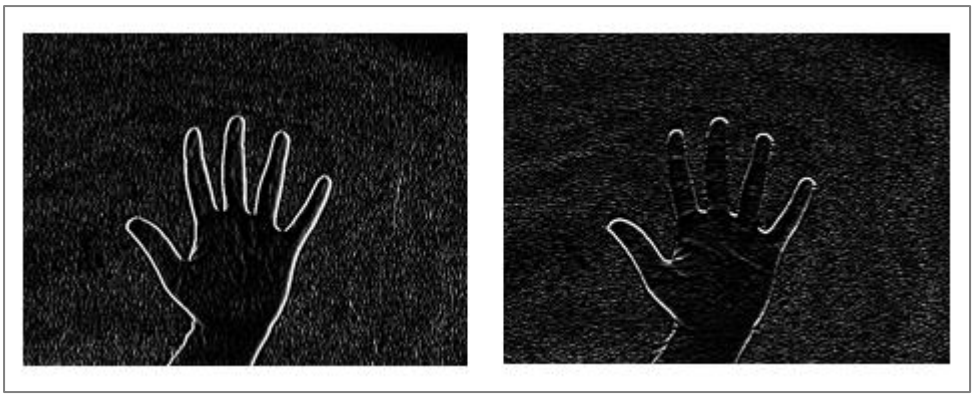

Ilustración 4.1 - Resultado de las derivadas de Sobel en x (derecha) e y (izquierda)

La *binarización de la imagen* es un proceso muy sencillo en el que se recorren de nuevo todos los píxeles y aquellos que estén por encima de un valor mínimo de variación de intensidad (umbral) se truncarán a blanco (255), o a negro (0) en caso contrario. De este modo se facilita el siguiente paso de búsqueda de contornos.

En la ilustración 4.3 se muestra el valor de variación de intensidad normalizado en escala de grises (izquierda) y su binarización (derecha). En ésta última se puede observar ruido, o puntos blancos que no pertenecen al contorno de la mano, a lo largo de todo el fondo. Esto es debido a que la elección del umbral es muy delicada y un valor muy elevado, que suprime el ruido, puede producir que se corte el contorno debido a brillos o sombras intensas en la imagen. Como se explicará posteriormente el efecto del ruido es suprimido usando ciertos criterios sobre la longitud del perímetro de los contornos hallados.

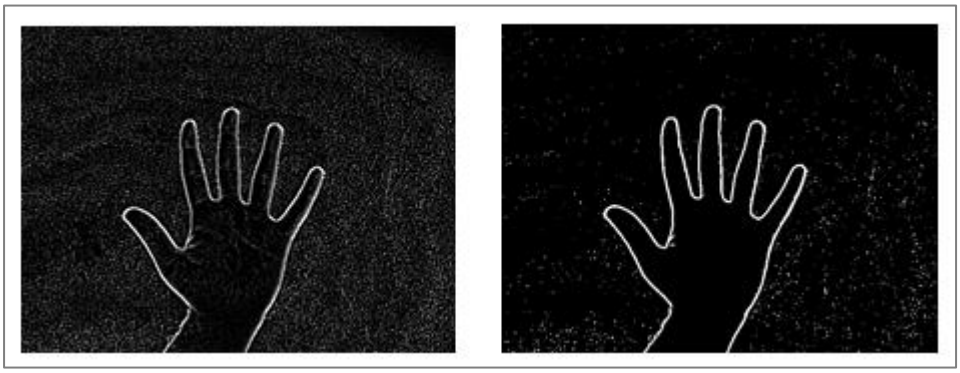

Ilustración 4.2 - Variación de intensidad normalizada en escala de grises (izquierda) y su binarización (derecha)

Cabe destacar que puesto que no tenemos una mano aislada, si no que en la imagen aparece también el brazo, los bordes obtenidos por el cálculo de la variación de gradientes no será un contorno cerrado. Para poder tener un contorno aproximado, únicamente de la mano, éste se completará artificialmente por debajo trazando una línea horizontal situada

una distancia proporcional a la longitud del lado del área de interés por encima del punto más bajo de ésta.

Por último se procede con la *búsqueda de contornos*. La búsqueda se realiza sistemáticamente sobre cada píxel de la imagen. En caso de encontrar un píxel objetivo (valor mayor que 0) se guarda y se continúa analizando sus vecinos (8-conectados, es decir, en cruz y en aspa). De este modo se van guardando únicamente los puntos contiguos, externos.

Después de hallar los diferentes contornos se hace un nuevo filtrado eliminando los contornos que sean internos a otros.

Todo este proceso se consigue aplicando el algoritmo Suzuki (1985)[5] cuya implementación excede el propósito de esta memoria y es utilizado en la función `findContours(binary,contours,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_SIMPLE)` donde:

- **binary:** parámetro de entrada. Imagen binaria con los puntos pertenecientes a posibles contornos de la que se han eliminado aquellos punos que caen fuera de la region de interés.
- **contours:** parámetro de salida. Vector donde se guardarán los contornos (vecotres de puntos) encontrados.
- **search_type:** parámetro de entrada. Opción de búsqueda. Se ha empleado **CV_RETR_EXTERNAL** que no tendrá en cuenta los contornos internos.
- **contour_type:** parámetro de entrada. Opción del tipo de contorno. Se ha utilizado **CV_CHAIN_APPROX_SIMPLE** a la hora de crear el contorno que en el caso de encontrar líneas verticales, horizontales o diagonales solo guardará los puntos de principio y fin de la línea. De este modo se mejora el tiempo de computación posterior al realizar cálculos sobre el contorno.

Para agilizar los cálculos y descartar contornos no pertenecientes a la mano, la búsqueda de contornos se realiza únicamente sobre una superficie circular inscrita en la región de interés proporcionada inicialmente.

Finalmente se recorrerán todos los contornos eliminando aquellos cuyo valor de perímetro no esté dentro de un rango de valores posibles calculado de forma experimental.

En la ilustración 4.4 se observa el contorno, relleno para facilitar la visualización, en color azul y en rojo la región circular sobre la que se buscan contornos.

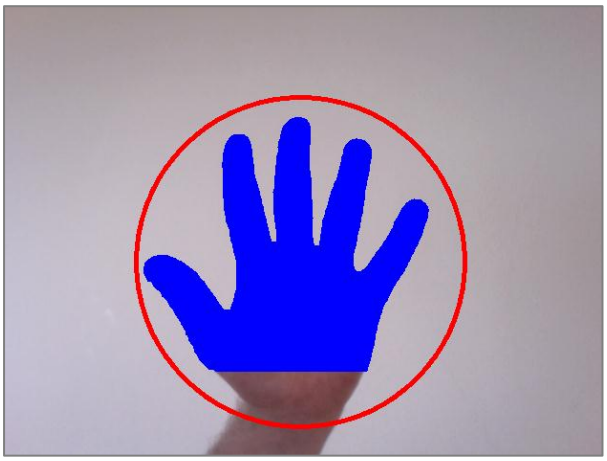

Ilustración 4.3 - Región circular y contorno

4.3. Cálculo de puntos estructurales

Un modo eficiente y robusto para obtener los puntos estructurales de la mano consiste en calcular un recubrimiento convexo del contorno de la mano y compararlo con el contorno.

El *cálculo del polígono de recubrimiento convexo* consiste en obtener un polígono cuyos lados unen los puntos más externos del contorno eliminando las concavidades, o *defectos de convexidad*, que éste pueda tener. Se realiza mediante técnicas convencionales de geometría computacional. En nuestro caso se ha realizado con la función `convexHull(contour,hux)` siendo:

- **contour:** parámetro de entrada. Vector de puntos del contorno sobre el que se quiere calcular el recubrimiento.
- **hull:** parámetro de salida. Vector de puntos del recubrimiento convexo o cáscara que se obtiene sobre el contorno.

La *obtención de los defectos de convexidad* permite identificar los espacios interdigitales, además de otras concavidades generadas por la forma de la mano o errores en el cálculo del contorno. Los defectos se van a representar, como se ha indicado anteriormente con su punto de inicio, su punto final y su punto de máxima profundidad.

Para su cálculo se recorre el contorno hasta encontrar el primer punto que difiere del recubrimiento. Este será el punto de inicio del defecto. Se siguen recorriendo los puntos del contorno hasta encontrar uno que coincida con el siguiente punto de la cobertura que será el punto final. A su vez se calculan las distancias de cada uno de los puntos del contorno con la recta que pasa por el punto de inicio y final. Aquel punto cuya distancia sea máxima es guardado como punto de máxima profundidad.
Este proceso se consigue con la ayuda de la función de OpenCV `convexityDefects(contour,hull,defects)` siendo:

- **contour:** parámetro de entrada. Vector de puntos del contorno.
- **hull**: parámetro de entrada. Vector de puntos del recubrimiento convexo.
- **defects:** parámetro de salida. Vector de defectos de convexidad con su punto de inicio, final y de máxima profundidad.

En la ilustración 4.5 se muestra el recubrimiento (en rojo) obtenido sobre el contorno (en azul). También se muestran los defectos de convexidad (en morado) con sus puntos de inicio y final (en verde) y los puntos de máxima profundidad (en amarillo).

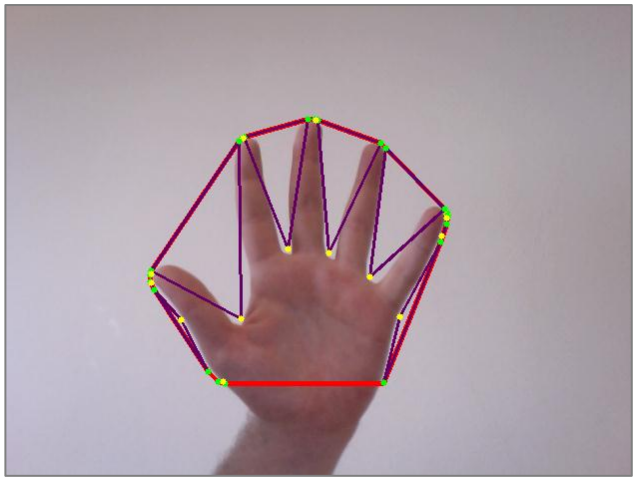

Ilustración 4.4 - Recubrimiento del contorno y defectos de convexidad

La obtención de los puntos de los extremos de los dedos y los espacios interdigitales se consigue con los siguientes pasos:

De todos los defectos encontrados se mantienen los cuatro defectos que tengan los puntos de máxima profundidad a mayor distancia. Estos deben de coincidir con los cuatro espacios entre los cinco dedos de la mano. Se eliminan por tanto defectos ocasionados por errores durante la búsqueda del contorno o posturas de la mano con respecto al comienzo del brazo. Los puntos de los espacios interdigitales serán los puntos de máxima profundidad.

Los puntos de los extremos de los cuatro primeros dedos coincidirán con los puntos iniciales de los cuatro defectos y el punto extremo del último dedo con el punto final del cuarto defecto.

Para eliminar posibles falsos positivos de defectos causados por sombras y brillos que superen el filtrado por mínima profundidad, se utiliza el valor del punto de inicio del defecto situado en uno de los extremos (mayor o menor valor de x) situado más abajo (menor valor de y). Con este valor se traza una línea imaginaria horizontal, *línea límite de defectos de convexidad*, que delimita la altura mínima a la que se deben encontrar los puntos de inicio y de fin de defectos válidos.

En la ilustración 4.6 se muestran en rojo los puntos de los extremos de los dedos, en azul los puntos interdigitales y en verde el baricentro de estos últimos.

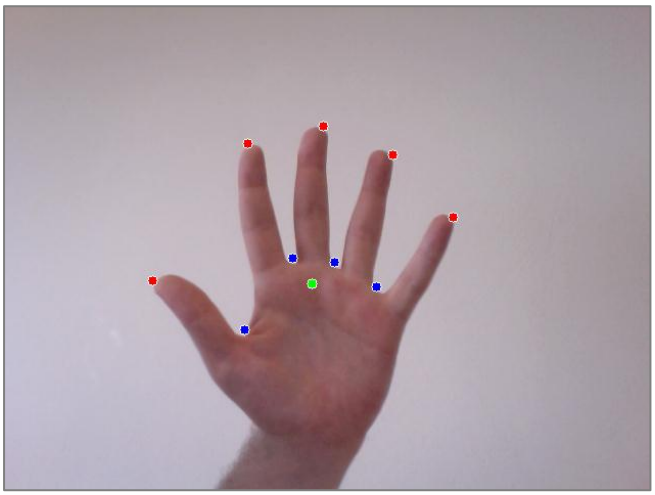

Ilustración 4.5 - Puntos estructurales

Las *condiciones* que se deben cumplir para que los puntos definan las características de una mano son:

- La mano debe estar en posición vertical para poder definir si se trata de una mano derecha o izquierda.
- Hay un mínimo de 4 defectos de convexidad.
- Todos los defectos de convexidad tienen su punto de máxima profundidad a una profundidad mínima.
- Si el punto extremo del primer dedo está por debajo del punto extremo del quinto se trata de la mano derecha.
- Si el punto extremo del primer dedo está por encima del punto extremo del quinto se trata de la mano izquierda.

- Si se trata de la mano derecha todos los puntos extremos de los dedos están por encima del primero
- Si se trata de la mano izquierda todos los puntos extremos de los dedos están por encima del último.

4.4. Resultados de tiempo

Las pruebas de la etapa de obtención de características se realizaron sobre un computador con las siguientes características:

- **Procesador:** Intel® Core™ i5 430M (2.27 GHz, 3 MB L3 cache)
- **RAM:** 4,00 GB DDR3
- **GPU:** ATI Mobility Radeon™ HD 5650 (Hasta 2736 MB HyperMemory™)

Los tiempos obtenidos en la ejecución de esta etapa oscilan entre los 13 ms y los 23 ms. Este tiempo es muy bajo comparado con los tiempos de la etapa anterior y no representa un valor significativo en el tiempo de ejecución global. Tomando el peor valor obtenido somos capaces de realizar esta etapa unas 43 veces por segundo, es decir, casi el doble de los frames por segundo que es capaz de detectar el ojo humano (20~30 fps). Esto nos permite utilizar este método en combinación con el método de seguimiento de puntos para dar robustez al algoritmo en la etapa siguiente.

4.5. Dificultades encontradas

En la fase del cálculo del contorno se han encontrado problemas a la hora de diferenciar la mano del resto de objetos de la imagen. Con una primera aproximación se intentó filtrar la imagen por la gama de colores de sus puntos, pero debido a que cualquier punto perteneciente a la cara o al brazo superarían este tipo de filtrado se optó por un algoritmo de reconocimiento por aprendizaje que acotara la zona donde se detecta una mano (región de interés). Gracias a que la zona es acotada, es posible aplicar un algoritmo de búsqueda de bordes basado en los valores máximos de los gradientes de intensidad horizontal y vertical de la imagen.

Además, como ya se ha descrito en los apartados anteriores, los bordes obtenidos tras la fase del cálculo de los gradientes de intensidad no proporcionaban un contorno cerrado debido a la aparición del inicio del brazo. La solución fue acotarlo artificialmente trazando una línea horizontal.

En esta fase también se presentó el problema de confundir el fondo con ciertos puntos de brillo de la mano. La solución en este caso ha sido la de ecualizar el histograma de la imagen consiguiendo un mayor contraste. Aun así permanece el problema de que en caso de que el fondo no sea uniforme, o existan brillos y sombras muy acentuadas, se puedan encontrar falsos positivos en el cálculo del borde de la mano.

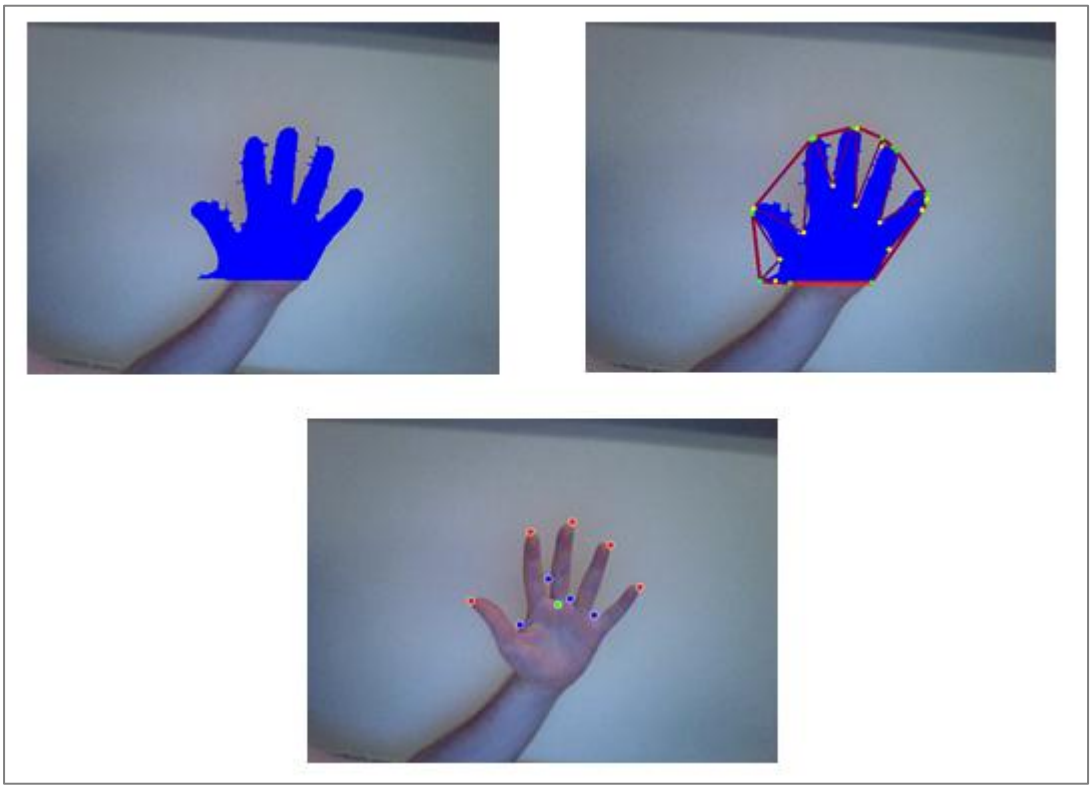

Ilustración 4.6 - Impacto de sombras en la resolución del contorno

El impacto de la influencia de zonas de brillos y sombras muy acentuadas, que dificultan el cálculo de un contorno regular, ha sido atenuado gracias al filtrado de los contornos obtenidos por un valor máximo y mínimo de su perímetro en combinación con la comparación de su recubrimiento convexo. De este modo, aunque se obtenga un contorno irregular que se introduzca o exceda la región de la mano, se siguen pudiendo identificar sus puntos estructurales de un modo aceptable fijándonos únicamente en los cuatro defectos de convexidad más profundos.

5. Seguimiento

Llegados a este punto del algoritmo, se dispone de todas las características necesarias de la mano y se procede a realizar un seguimiento de los puntos de interés de la misma, i.e. los extremos de los dedos. Para conseguir este objetivo se usa el método de *Lucas-Kanade*[6], una técnica de estimación de flujo óptico que, partiendo de la suposición de que cada píxel de una secuencia de vídeo se va moviendo a píxeles vecinos en un cierto rango, consigue reconocer los puntos de interés en cada nuevo frame. Para aplicar este método se debe disponer siempre de dos frames consecutivos, de forma que a partir de la posición de los puntos en el frame antiguo se pueda calcular su posición en el frame nuevo.

Este proceso de seguimiento se divide entonces en tres etapas: una primera de inicialización o pre-procesamiento, una segunda encargada de aplicar el método de *Lucas-Kanade* y una tercera que se encargará de actualizar las características de la mano. El flujo en que se desarrolla esta fase queda bien reflejado en la ilustración

5.1. Inicialización

En esta etapa se almacena la imagen en escala de grises del primer frame que contiene una mano estable así como las características asociadas a ella, obtenidas previamente en la fase anterior. Estos servirán de punto de partida para comenzar el seguimiento con el siguiente frame.

5.2. Seguimiento del flujo óptico

A partir de un frame de entrada y los puntos de los dedos almacenados anteriormente, en esta etapa se consigue reconocer la posición de dichos puntos en el nuevo frame siguiendo su flujo óptico.

5.2.1 Flujo óptico

El flujo óptico es el patrón de movimiento aparente de los bordes en una escena causado por el movimiento relativo entre un observador, en este caso la cámara, y la escena. Para su estimación se requiere de una secuencia de frames debidamente ordenados.

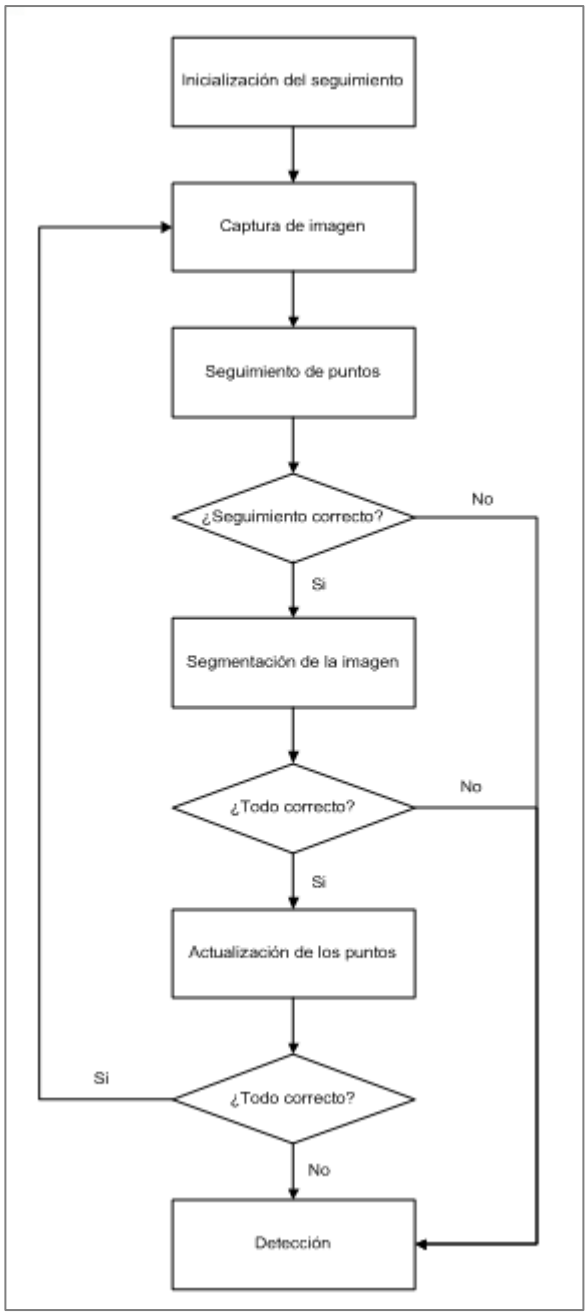

Ilustración 5.1 - Diagrama de flujo de la fase de seguimiento

Los métodos de flujo óptico tratan de calcular el movimiento entre dos imágenes que fueron tomadas en los momentos de tiempo t y $t + \Delta t$. Estos métodos utilizan derivadas parciales con respecto a las coordenadas espaciales y temporales.

Dado un punto asociado a un píxel (x,y) en la imagen tomada en el instante t, se define $I(x,y,t)$ como la intensidad del píxel en esa imagen. El mismo punto en la imagen tomada en el instante $t + \Delta t$, tendrá asociado un píxel $(x + \Delta x, y + \Delta y)$, siendo Δx y Δy la variación de posición con respecto al píxel anterior, y la intensidad asociada a este nuevo

píxel es igual a:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{5.1}$$

El problema del flujo óptico considera la siguiente restricción:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{5.2}$$

Desarrollando las series de Taylor sobre (5.1) se obtiene la siguiente expresión:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t \tag{5.3}$$

Y debido a la restricción (5.2) se tiene que:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0 \tag{5.4}$$

Dado que el flujo óptico considera movimiento de objetos, se puede establecer la velocidad con que se mueve un punto como la relación existente entre la variación de su posición en ambos ejes con respecto a la variación de tiempo Δt. Entonces, se define

$$V_x = \frac{\Delta x}{\Delta t} \;\; y \;\; V_y = \frac{\Delta y}{\Delta t} \tag{5.5}$$

como la velocidad de ese punto en los ejes x e y respectivamente.

Dividiendo la expresión (5.4) entre la variación de tiempo Δt y aplicando (5.5) se tiene:

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = \frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \tag{5.6}$$

Renombrando las derivadas parciales como $I_x = \frac{\partial I}{\partial x}, I_y = \frac{\partial I}{\partial y}, I_t = \frac{\partial I}{\partial t}$, se tiene finalmente la siguiente ecuación:

$$I_x V_x + I_y V_y = -I_t \tag{5.7}$$

Dado que esta ecuación contiene dos incógnitas, V_x y V_y, no podrá ser resuelta de forma directa. Para hallar una solución será necesario añadir alguna restricción adicional.

5.2.2 Método de *Lucas-Kanade*

Bruce D. Lucas y Takeo Kanade proponen una restricción para resolver problemas de flujo óptimo: considerar que el desplazamiento de un píxel entre dos frames cercanos en tiempo es pequeño y aproximadamente constante. Por ello, se puede establecer una ventana de búsqueda en torno al píxel que se desea perseguir.

Entonces, dado un píxel cuyo vector de movimiento es (V_x, V_y), éste debe satisfacer:

$$\begin{aligned} I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\ &\vdots \\ I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n) \end{aligned}$$

donde $q_1, q_2, \ldots, q_n$ son los píxeles que están en el interior de la ventana de búsqueda e $I_x(q_i)$, $I_y(q_i)$ y $I_t(q_i)$ son las derivadas parciales de la intensidad del píxel q_i en x, y y t respectivamente.

Se pueden expresar estas ecuaciones en forma de matriz $Av = b$ obteniéndose lo siguiente:

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

Dado que este sistema contiene más ecuaciones que incógnitas, para calcular una solución factible se empleará la aproximación matemática de los mínimos cuadrados.

La función de OpenCV `calcOpticalFlowPyrLK()` implementa el método de *Lucas-Kanade* y, además, lo optimiza realizando un seguimiento de los puntos sobre distintos tamaños de la misma imagen realizando progresivas reducciones piramidales[7]. Por ejemplo, realizando una reducción piramidal de tres niveles como la que se muestra en la ilustración 5.1, los niveles más bajos (L_2) permiten reconocer movimientos más rápidos y bruscos, ya que aplicando un mismo tamaño de ventana en éstas se comprueban puntos cuya distancia real es mayor. Tras esto, se procede a comprobar en los niveles altos (L_0) ya que estos proporcionan una mayor precisión de los cálculos.

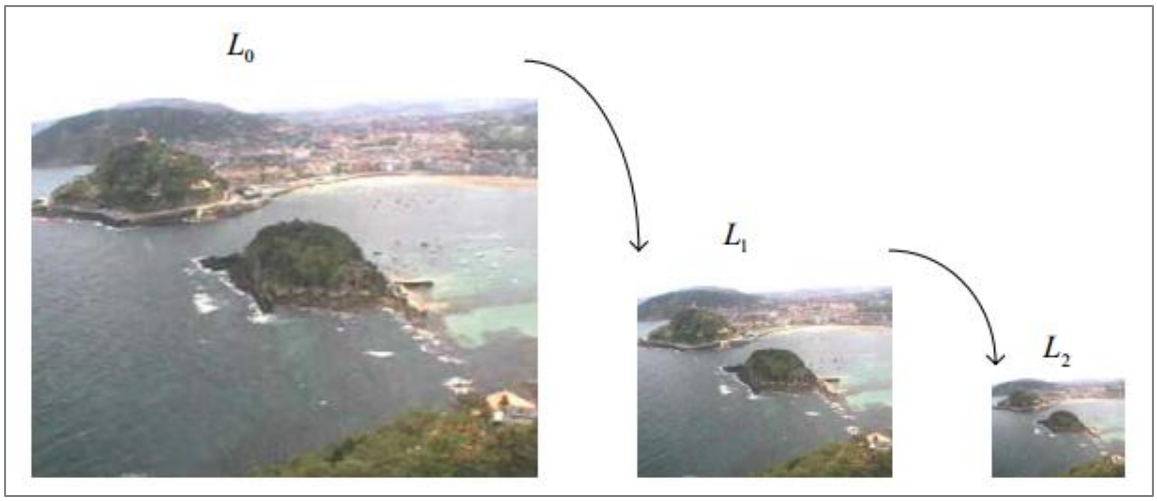

Ilustración 5.2 - Reducción piramidal de tres niveles

5.3. Actualización de los puntos

A partir de los puntos recibidos tras aplicar el método de *Lucas-Kanade* y de la mano calculada en el frame anterior, esta etapa calcula las características de la mano en el frame actual. También se detectará que dedos se han flexionado y cuales se han estirado calculando los defectos de convexidad de la nueva mano y comparándolos con los puntos devueltos en la segunda etapa del seguimiento.

El primer paso será obtener los puntos estructurales de la nueva mano como ya se hizo en la etapa de obtención de características: se calcula la imagen de magnitudes de gradiente, se binariza, se aplica la región de interés a la imagen binarizada, se inserta una línea por debajo de la mano para cerrar el contorno, se calcula el contorno, el envoltorio de ese contorno y los defectos de convexidad de ese envoltorio, se establece la línea límite de defectos válidos para considerar que contienen posibles extremos de los dedos y se seleccionan aquellos cuatro cuya profundidad sea máxima.

Sin embargo, esta etapa se diferencia en cuatro aspectos de la fase anterior. La primera de ellas es que no se dispone de una región facilitada por la fase de detección por lo que se debe buscar otra forma de obtenerlo. En este caso se ha optado por usar la región de la mano del frame anterior añadiéndole un cierto margen a los cuatro lados. Aplicando esta solución se obtienen buenos resultados ya que la diferencia de posición de la mano entre dos frames consecutivos no es muy alta por lo que esta nueva región consigue contener la mano entera. Cabe tener en cuenta que esto también acarrea una limitación: movimientos muy rápidos no podrán ser seguidos ya que ello implicaría que la nueva región probablemente cortaría la

mano o incluso podría llegar a no contener ninguna. Aun así, se asumirá esta limitación.

Una segunda diferencia es que, mientras en la anterior fase se puede insertar una simple línea horizontal en la parte inferior de la región para cerrar el contorno gracias a que la mano debe de estar en posición vertical, en esta etapa podemos tener la mano girada a la izquierda o a la derecha sobre el eje z. Por ello, no podemos dibujar una línea horizontal en la parte inferior ya que ello podría cortar la mano. La solución: dibujar una línea en la parte inferior de la mano cuya inclinación es igual a la normal del vector que indica la inclinación de la mano (Ilustración 5.2). Esta forma de insertar la línea supone considerar que entre frames consecutivos el cambio de ángulo de la mano no es muy grande.

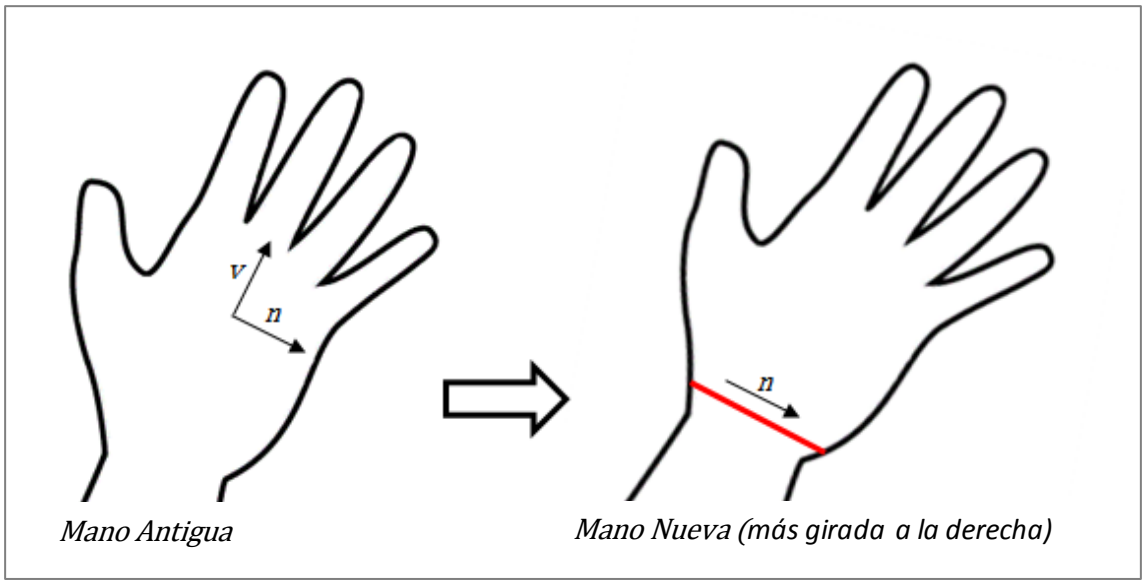

Ilustración 5.3 – Línea inferior de cierre de contorno (en rojo)

La tercera se da en la forma de establecer la línea límite de defectos de convexidad válidos. Mientras que en la fase de obtención de características se establecía a partir de los puntos más extremos tanto a izquierda y derecha del contorno por estar dicha mano en posición vertical, en este nuevo caso la mano podría estar girada por lo que no se puede realizar esa aproximación. Sin embargo, gracias a los puntos obtenidos por el método de *Lucas-Kanade* se sabe donde se encuentran los puntos asociados a los dedos pulgar y meñique por lo que se puede tomar la línea que forman estos dos puntos para establecer la línea límite, añadiéndole un cierto margen para evitar desechar defectos que se encuentren en la frontera.

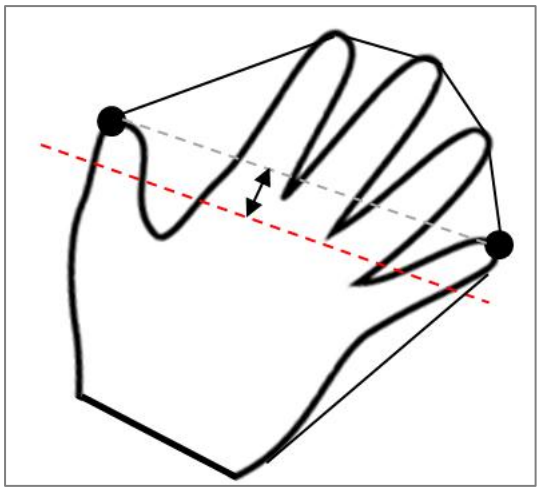

Ilustración 5.4 - Línea límite de defectos de convexidad factibles (en rojo)

La cuarta y última de las diferencias se da en la forma de obtener la punta de los extremos de los dedos. En la fase anterior se realizaba haciendo un recorrido a lo largo del eje x para ir asignando un punto a cada dedo, en esta etapa la mano además de poder estar girada también puede tener algún dedo flexionado por lo que tampoco se puede aplicar el método usado anteriormente. Pero partiendo de los puntos obtenidos por el método de *Lucas-Kanade* se pueden obtener el nuevo punto extremo de cada dedo buscando aquel inicio o fin de defecto de convexidad que se encuentren más próximos a él en un cierto rango. De no encontrarse, eso significaría que ese dedo se flexionó y se deberá marcar ese dedo como flexionado en las características de la nueva mano. Para los dedos que se encontraban flexionados en la mano antigua, y por tanto no se obtuvo ningún punto por flujo óptico del mismo, se marcará como estirado cuando, comprobando con el defecto del dedo anterior del dedo flexionado, el inicio del defecto asociado al dedo anterior no sea el dedo siguiente al flexionado. Esto limita a una única flexión de dedo al mismo tiempo. Además, no se podrán flexionar los dedos pulgar y meñique ya que su flexión no provoca una desaparición de defecto.

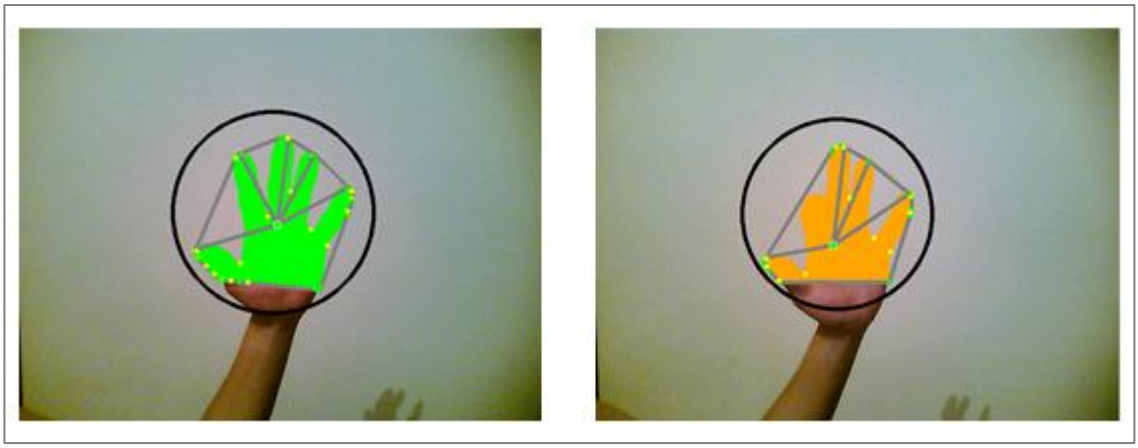

Ilustración 5.5 - Estiramiento y flexión de dedo

Es además en este punto donde se calculan los vectores de movimiento asociados a cada uno de los extremos de los dedos. Esto se realiza restándole a cada nuevo punto calculado su valor en la antigua mano.

5.4. Resultados de tiempo

Con éste método se consigue reducir el tiempo de búsqueda de la posición de los dedos.

Ejecutando el programa sobre un portátil de las siguientes características:

- **CPU:** Intel Atom Processor N450 (1,66 GHz, 512KB cache)
- **Memoria RAM:** 1GB
- **Tarjeta Gráfica:** Intel Graphics Media Accelerator 3150
  - **Pipelines:** 2
  - **Velocidad núcleo:** 200MHz
  - **Memoria de gráficos:** 250MB

Se obtienen unos tiempos de ejecución de esta parte del algoritmo que oscilan entre los 40 ms y los 100 ms, lo cual supone una reducción importante del tiempo, para una máquina cuyas prestaciones gráficas y de computación son bastante pobres.

Realizando las mismas pruebas en un ordenador con mejores prestaciones:

- **Procesador:** Intel® Core™ i5 430M (2.27 GHz, 3 MB L3 cache)
- **RAM:** 4,00 GB DDR3
- **GPU:** ATI Mobility Radeon™ HD 5650 (Hasta 2736 MB HyperMemory™)

los tiempos obtenidos se encuentran entre los 30 ms y los 35 ms lo cual supone una notable mejora de rendimiento comparado con los resultados del anterior computador.

5.5. Dificultades encontradas

En un principio, el seguimiento se realizó basándose única y exclusivamente en el flujo óptico. Ello provocaba que se perdiesen los puntos con bastante frecuencia, sobretodo al realizar una flexión con el dedo ya que al estirar después no era capaz de continuar siguiendo el punto. No se encontraba una solución clara a este problema y no surgían muchas alternativas para su resolución.

Una de ellas fue eliminar por completo el uso del método de *Lucas-Kanade* y realizar un seguimiento de la mano utilizando únicamente la detección. Esta propuesta tiene algunos puntos débiles: es muy difícil reconocer la posición de los nuevos dedos y el tiempo de detección no es lo suficientemente rápido como para hacerlo de forma continuada frame a frame (una detección aplicada a una región determinada tarda del orden de 100ms lo que provoca un ratio de captura de solamente 10fps). Por todo ello, esta opción fue desechada.

Finalmente, con la resolución tomada usando comparación de puntos de flujo óptico y de defectos de convexidad se ha conseguido un seguimiento mucho más sólido en el que la perdida de puntos se produce muy raramente y donde el retorno de flexión de un dedo se realiza de forma automática.

6. Interpretación

La etapa de interpretación se encarga de reconocer gestos y realizar acciones cuando éstos se produzcan. Para poder llevar a cabo esta tarea necesita partir de una mano con todas sus características calculadas, ya que es con esas características con las que se va a interpretar su gesto.

Un gesto se puede estar formado por los siguientes movimientos: un desplazamiento de la mano, un giro, flexiones de los dedos, o la combinación de ellos. Por tanto para identificar cuál es el gesto que está adoptando la mano, se tiene que identificar si se está realizando, o no, cada uno de los movimientos anteriores.

En esta etapa de interpretación se realiza la interacción con la computadora, hasta este momento, las etapas anteriores preparaban los datos para facilitar la interpretación.

La interacción se realiza mediante el lanzamiento de eventos de teclado (`INPUT_KEYBOARD`)[1], o de ratón (`INPUT_MOUSE`)[1]. El gesto que en cada momento está realizando la mano es el que decide que evento es el que se va a lanzar, ya que cada gesto tiene asociado un único evento.

La interacción mediante el lanzamiento de eventos permite que se pueda interactuar con otras aplicaciones mientras que se tiene *HandI* (la aplicación desarrollada) ejecutándose en segundo plano.

6.1. Desplazamiento de la mano

La mano se puede mover en las tres dimensiones del espacio, en el eje horizontal (hacia izquierda o derecha), en el eje vertical (hacia arriba o abajo) y en eje de profundidad (acercándose o alejándose de la cámara).

Aunque la mano tiene gran variedad de movimientos, sólo se van a considerar como gestos válidos, aquellos movimientos que se produzcan en los ejes horizontal y vertical. Los movimientos producidos en el eje de profundidad, son reconocidos por el algoritmo pero no son gestos.

[1] Las constantes INPUT_KEYBOARD e INPUT_MOUSE definen el valor de la máscara para señales de entrada/salida procedentes de teclado o ratón en el Sistema Operativo Windows

Al reconocer sólo gestos en dos dimensiones, se puede establecer una restricción: *Una mano se mueve a lo largo de un eje si y solo si el movimiento de todos sus dedos se realiza en el mismo eje.*

Cada uno de los dedos de la mano tiene asociado como característica el vector de movimiento, que es calculado en las últimas fases de la etapa de seguimiento. A partir de este vector podemos conocer hacia donde se están desplazando cada uno de los dedos en función del signo de sus componentes.

Los movimientos de los dedos quedan restringidos a dos direcciones distintas, y por tanto en cuatro sentidos distintos, que se corresponden con los ejes horizontal y vertical. Por ejemplo si la mano se desplaza hacia arriba se toma como movimiento vertical hacia arriba (GO_UP[2]), y como movimiento horizontal un valor que identifica que no se produce movimiento en ese eje (GO_NOWHERE[2]).

Los movimientos compuestos, no se considerarán de forma independiente, sino que se establece que su movimiento se realiza en todos los sentidos de los que se compone. Por ejemplo: Si una mano se desplaza hacia arriba y hacia la derecha, se toma como movimiento vertical hacia arriba (GO_UP[2]) y como movimiento horizontal hacia la derecha (GO_RIGHT[2]).

Se dispone de una constante, llamada CLOSE_TO_AXE, que toma un valor cercano a cero, que se utiliza para identificar cuando una componente del vector de movimientos está cerca del eje o no. Se necesita esta constante ya que el seguimiento puede desplazar unos pocos píxeles la posición de los puntos de los dedos de un frame a otro, originando un vector de movimiento que no queremos considerar como movimiento válido al ser muy pequeño.

El valor asignado al movimiento, se realiza en función del valor de las componentes normalizadas del vector de movimiento del punto que representa el dedo. De ésta forma se considera que un dedo se mueve hacia arriba (GO_UP) si el valor de la componente *y* de su movimiento es positiva y mayor que la constante CLOSE_TO_AXE. Mientras que se considera que un dedo se mueve hacia abajo (GO_DOWN) si la componente *y* de su movimiento es negativa y menor que el valor negativo de la constante. Un dedo se considera

[2] Las constantes que identifican los desplazamientos que realizan los dedos, toman el nombre de su voz inglesa, así GO_UP, GO_DOWN, GO_ LEFT, GO_ RIGHT representan los desplazamientos hacia arriba, abajo, izquierda y derecha respectivamente. Mientras que GO_NOWHERE indica que no se produce movimiento en ninguno de los dos ejes.

que ha permanecido estático (GO_NOWHERE) si la componente *y* de su movimiento se tiene un valor perteneciente al intervalo $[-CLOSE\_TO\_AXE, CLOSE\_TO\_AXE]$.

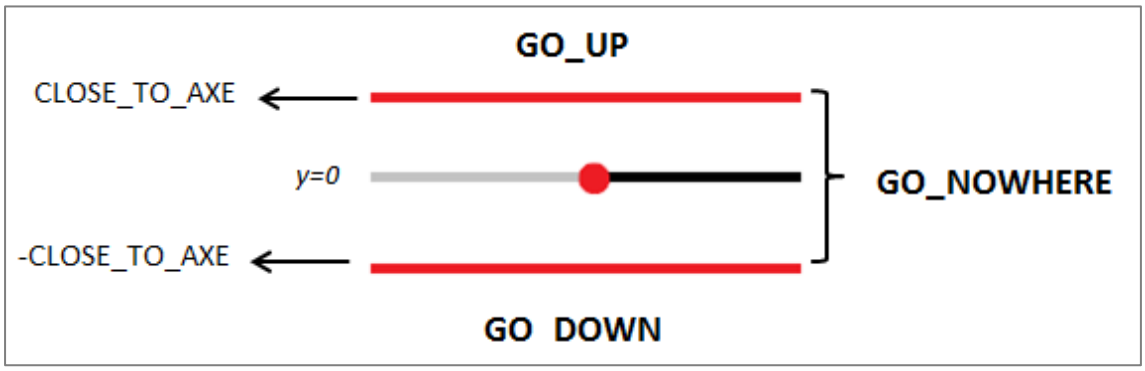

Ilustración 6.1 - Movimientos en el eje vertical

De una forma similar a la anterior se establece el movimiento de un dedo en el eje horizontal. Se considera que un dedo se mueve hacia la izquierda (GO_LEFT) si el valor de la componente *x* de su movimiento es positiva y mayor que la constante CLOSE_TO_AXE. Mientras que se considera que un dedo se mueve hacia abajo (GO_RIGHT) si la componente *x* de su movimiento es negativa y menor que el valor negativo de la constante. Un dedo se considera que ha permanecido estático (GO_NOWHERE) si la componente *x* de su movimiento se tiene un valor perteneciente al intervalo $[-CLOSE\_TO\_AXE, CLOSE\_TO\_AXE]$.

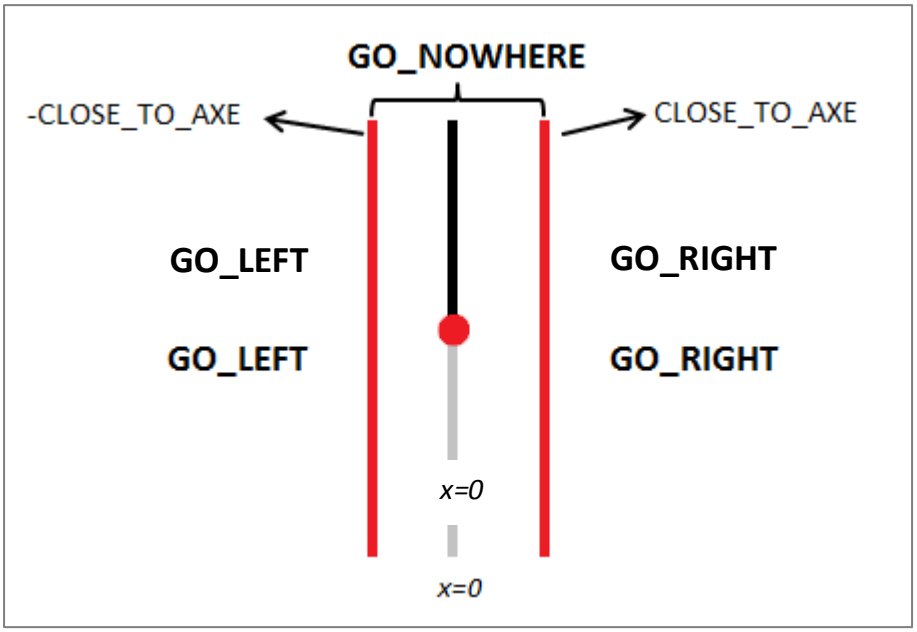

Ilustración 6.2 - Movimientos en el eje horizontal

El movimiento vertical de la mano es un determinado movimiento, a saber, arriba, o

abajo, si todos los dedos tienen asignado ese mismo movimiento, de lo contrario se establecerá que la mano no se mueve en ese eje.

El movimiento horizontal de la mano es un determinado movimiento, a saber, derecha o izquierda, si todos los dedos tienen asignado ese mismo movimiento, de lo contrario se establecerá que la mano no se mueve en ese eje.

6.2. Giro de la mano

Al igual que ocurría con el desplazamiento, la mano puede girar alrededor de los tres ejes del espacio. Pero a diferencia del desplazamiento, los giros tiene unos límites impuestos por la anatomía de la mano, la muñeca impide que la mano pueda girar libremente en todos los sentidos 360º.

En el caso del giro se dejan de reconocer los giros alrededor de los ejes vertical y horizontal del espacio debido a que dichos giros cambian la forma que tendría el contorno de la mano y por tanto harían que esos giros no fueran reconocidos por las etapas que obtienen las características de la mano.

Por tanto, se reduce a uno el giro que se reconoce. El giro alrededor del eje de profundidad, un giro que se puede identificar con inclinar la mano hacia la derecha o hacia la izquierda.

Para conocer la inclinación de la mano se recurre a otra de sus características calculadas durante el tracking, el ángulo de inclinación del dedo corazón, con respecto a su inclinación en la primera vez que se calculó.

Se define una constante, llamada MAX_VERTICAL_ANGLE, que toma un valor de ángulo cercano a cero que se establece como ángulo máximo para el cual la mano deja de estar en posición vertical y pasa a estar inclinada. Si el ángulo de la mano es mayor que esa constante, la mano estará inclinada hacia la izquierda. Mientras que, sí el ángulo de la mano es menor que el valor negativo de esa constante, la mano estará inclinada hacia la izquierda.

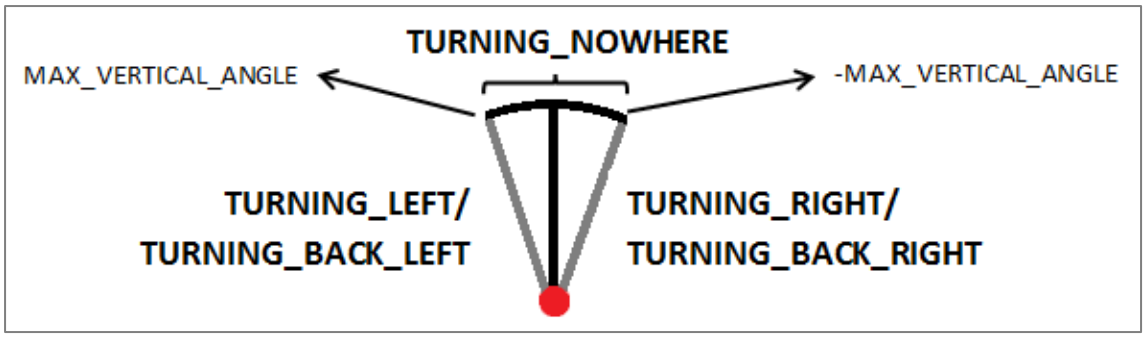

Ilustración 6.3 - Movimientos de giro

Sólo con esta información no se pueden conocer todas las características del giro. Falta por conocer una característica que permite distinguir más gestos de la mano, dicha característica es saber hacia donde se está dirigiendo el giro, si la mano está alejándose de la verticalidad o, por el contrario, está volviendo a ella. Para obtener está característica se utiliza, de nuevo, el vector de movimiento que representa el desplazamiento del dedo corazón.

De la misma forma que se hacía en el cálculo del desplazamiento de la mano, se utiliza la misma constante para saber si el movimiento del dedo ha sido suficiente para considerarlo como giro. En este caso realizamos la comparación con la componente *x* normalizada del vector, ya que esta componente siempre tendrá un valor distinto de cero, debido a los límites de inclinación que tiene la mano, que de forma natural no se inclina más de 90º hacia la derecha, ni hacia la izquierda. En función de su signo y del ángulo de la mano, este valor indicará si la mano está aumentando su inclinación o, por el contrario, está recuperando su posición vertical.

Sea α el ángulo de la mano, *x* la componente normalizada del vector del dedo corazón, θ la constante MAX_VERTICAL_ANGLE y *c* la constante CLOSE_TO_AXE, se define el movimiento de giro de la mano de la siguiente forma:

$$handTurnMovement = \begin{cases} TURNING\_RIGHT & si \ \propto < -\theta \ y \ x > c \\ TURNING\_BACK\_RIGHT & si \ \propto < -\theta \ y \ x < -c \\ TURNING\_LEFT & si \ \propto > \theta \ y \ x < -c \\ TURNING\_BACK\_LEFT & si \ \propto > \theta \ y \ x > c \\ TURNING\_NOWHERE & eoc \end{cases}$$

Donde TURNING_RIGHT y TURNING_LEFT son constantes que identifican que el giro se está realizando hacia derecha o izquierda, inclinándose cada vez más. Las constantes TURNING_BACK_RIGHT y TURNING_BACK_LEFT identifican el momento en el que

la mano está recuperando su verticalidad desde derecha o izquierda. La constante TURNING_NOWHERE indica que la mano está completamente vertical.

6.3. Flexión de los dedos

En el caso de las flexión de los dedos se tienen menos combinaciones de gestos, tan sólo cinco posibles flexiones, si tenemos en cuenta que la restricción que impide reconocer más de una flexión a la vez.

Una de las características de la mano nos índica cual es el dedo que se está pulsando en el momento en el que se realiza esta etapa del procesamiento, y se proceda a utilizar dicha información directamente para identificar que dedo de los cinco está realizando un clic.

6.4. Interpretación del movimiento y lanzamiento de eventos

Interpretar el movimiento que realiza el usuario, y relacionarlo con el lanzamiento de un evento, puede ser la parte más subjetiva de todo el proyecto, ya que se tienen que considerar que tipo de combinaciones de eventos se quieren tratar.

Para simplificar la interpretación de los gestos, se ha optado por realizar una implementación basada en máquinas de estados para los cuatro tipos de movimientos que se pueden realizar simultáneamente, los cuales ya se ha definido cómo se obtienen sus valores.

La máquina de estados para movimientos en el eje vertical, se activa si a ésta etapa le llega información de que se está moviendo la mano a lo largo de dicho eje.

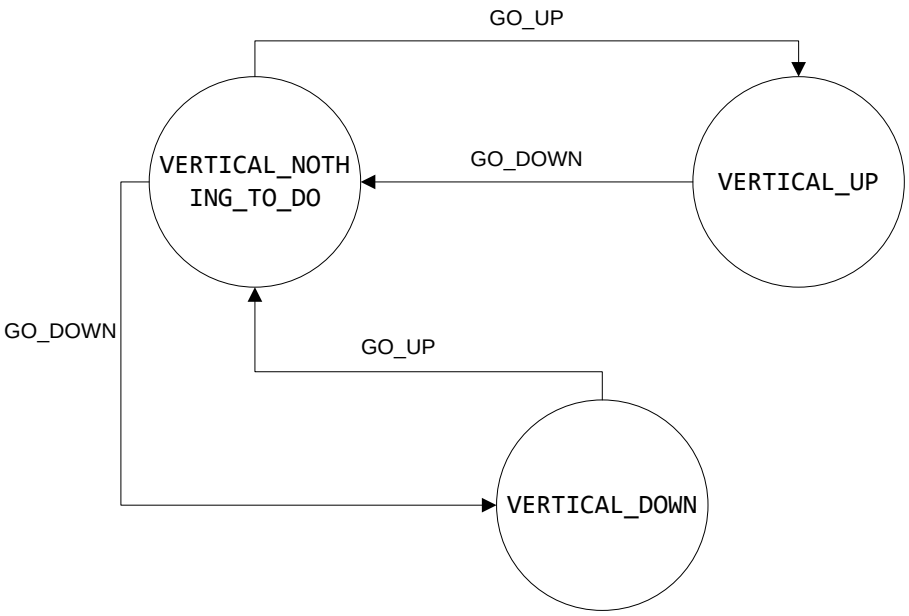

Ilustración 6.4 - Máquina de estados del movimiento vertical

Con que se produzca un movimiento hacia arriba, por ejemplo, se activa la máquina de estados, y permanece en este estado hasta que no se produce el primer movimiento contrario, en el caso del ejemplo, un movimiento hacia abajo, que lo cancela.

La máquina de estados para movimientos en el eje horizontal, se activa si a ésta etapa le llega información de que se está moviendo la mano a lo largo de dicho eje. Su funcionamiento es similar a la máquina que gestiona el eje vertical.

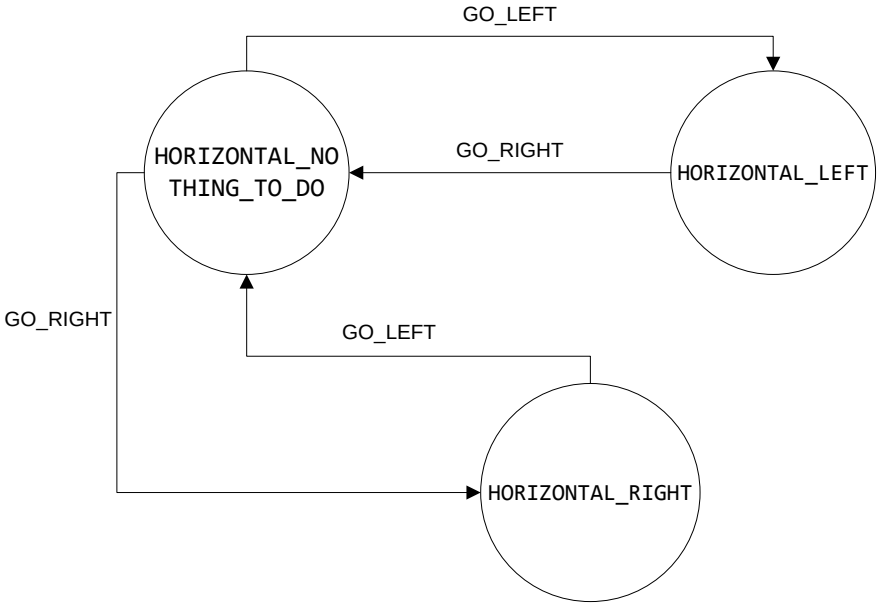

Ilustración 6.5 - Máquina de estados del movimiento horizontal

La máquina de estados para giros de la mano, se activa si a ésta etapa le llega información de que se está inclinando la mano hacia uno de los dos sentidos.

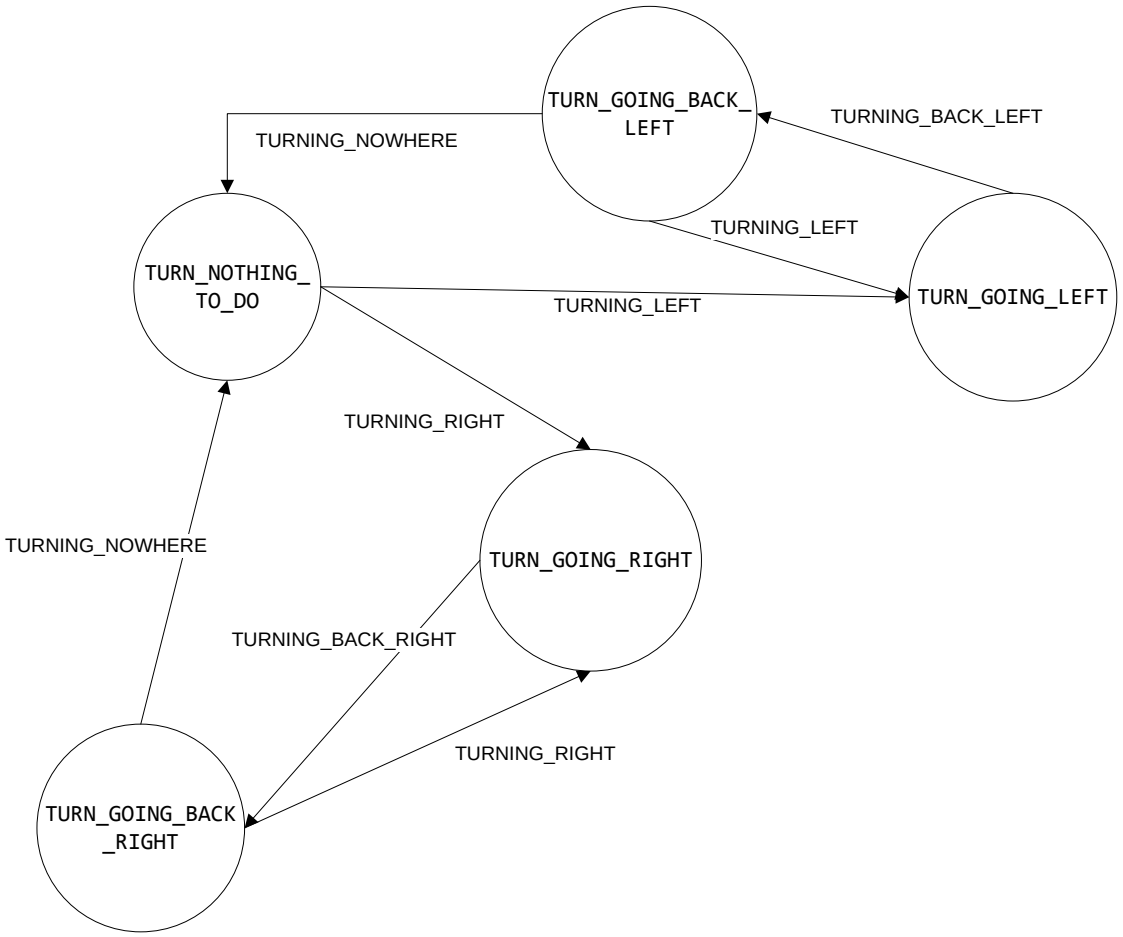

Ilustración 6.6 - Máquina de estados del movimiento de giro

En este caso, un giro solo se cancela en el momento en el que la mano recupera la verticalidad, pero se distinguen dos estados, uno para cuando la mano se está inclinando y otro para cuando está volviendo a su posición inicial, de esta forma se pueden asignar distintos eventos a estos movimientos que son distintos.

La máquina de estados que controla la flexión de los dedos, se activa si a ésta etapa le llega información de que un dedo está siendo flexionado.

Estas cuatro máquinas de estados, son independientes entre sí, y pueden funcionar a la par, pudiéndose dar una combinación de movimientos que como máximo va a llegar a ser: realizar un clic mientras la mano está girada, habiéndose desplazado a lo largo del eje horizontal y vertical.

La elección del evento a lanzar con cada movimiento se realiza mediante una configuración. La configuración define los estados que tienen que tener todas máquinas definidas anteriormente para lanzar el evento de dicha configuración, así como otros

parámetros que permiten ajustar cada cuantos frames se quiere que se repita el evento, el tipo de evento a realizar (evento de teclado o evento de ratón) y el evento concreto que se quiere que se realice (el código de una tecla o botón concreto).

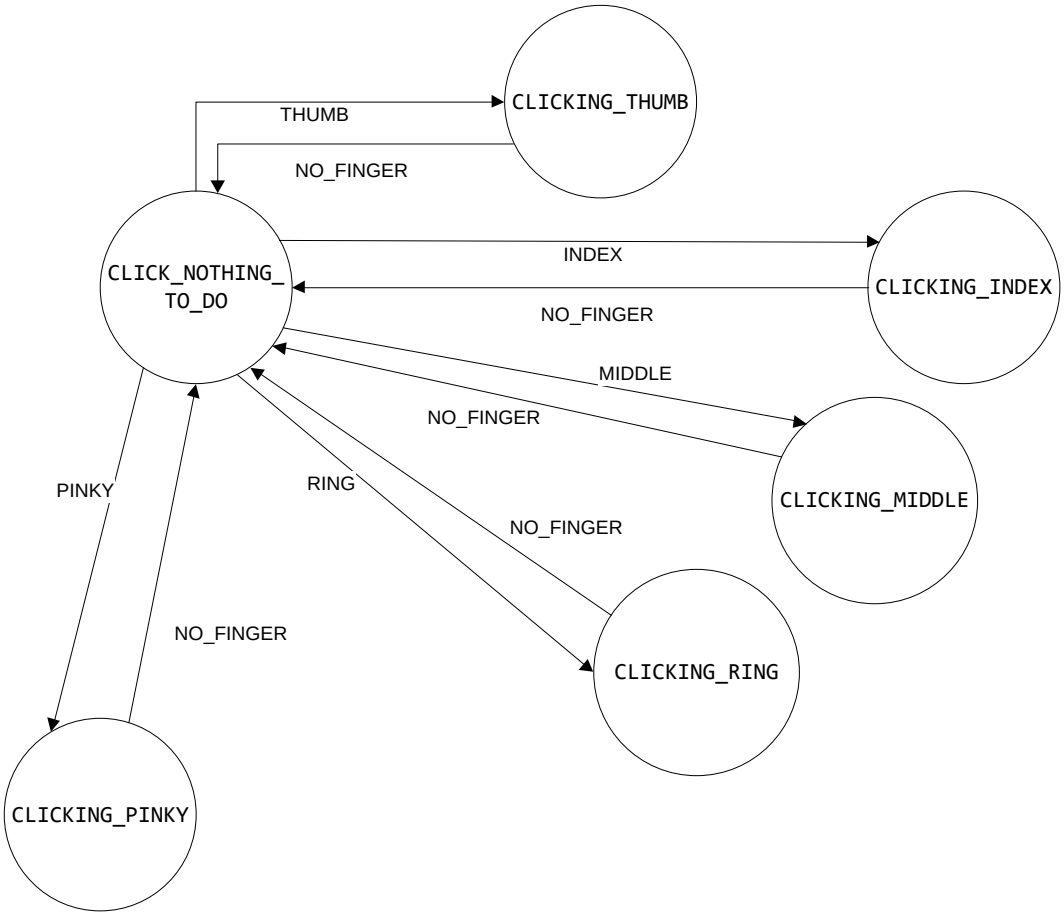

Ilustración 6.7 - Máquina de estados de identificación de clic

Tras actualizarse el valor de los estados en todas las máquinas se procede a buscar en la estructura que almacena las configuraciones, una de ellas que sea válida para ejecutar su evento. Una configuración es válida en el momento en el que los estados de las cuatro máquinas coincidan con los que tiene la configuración en sus parámetros, y en ese momento se lanzará el evento que tiene asociado dicha configuración.

El lanzamiento del evento se realiza mediante el paso de un mensaje al Sistema Operativo con la información del evento que se quiere generar, delegando así en el Sistema Operativo la gestión del envío del mensaje al lugar que corresponda, que en el caso de eventos de teclado y ratón, estos se envían a la aplicación que en ese momento esté en activa en la pantalla.

7. Implementación

Para la implementación de este método de interacción se eligió el lenguaje de programación *C++* y la librería de visión computacional de código abierto *OpenCV*. De esta librería ya se han comentado varias de las funciones que se utilizan a lo largo de este documento, las cuales son solo una pequeña porción de todas las herramientas que ofrece esta potente y a la vez compleja librería.

A la hora de crear el modelo que se utiliza para la auto-detección de manos, se optó por *SVM*^{light}, una implementación de código abierto de Máquinas de Soporte Vectorial.

El tiempo de procesamiento de cada frame afecta al ratio de captura de nuevos frames, incrementándose la diferencia en el desplazamiento de los objeto. Por esta razón se ha utilizado *qt-opencv-multithreaded* [8], una interfaz gráfica que dispone de un buffer en el que se almacenan hasta un máximo de 999 frames. En el buffer se va almacenando con un ratio de captura que le permite la capacidad restante del buffer, según se vaya procesando cada frame, éste dejará un hueco libre en el buffer. Esta implementación permite que las diferencias entre dos imágenes consecutivas sean menores que con el método implementado anteriormente.

La interfaz anterior es la que tiene el método principal que lanza la aplicación, y éste es el que se encarga de lanzar el algoritmo desarrollado para cada uno de los frames capturados. La implementación del algoritmo desarrollado sigue la siguiente estructura:

La clase principal es *HandInteraction*, que relaciona todas las demás clases y es la que se va ejecutando los pasos del algoritmo. También se encarga de almacenar cálculos intermedios para pasarlos de una clase a otra.

La clase *Hand* es la clase que identifica a una mano, almacena sus características específicas, y dispone de todas las funciones que permiten calcular dichas características

Se dispone de una clase general *Detector* que implementa funciones generales y comunes para realizar la detección de cualquier objeto en una escena.

La clase *HandDetector* hereda de *Detector* añadiendo funciones específicas para reconocer que el contorno que se detecta es el de una mano.

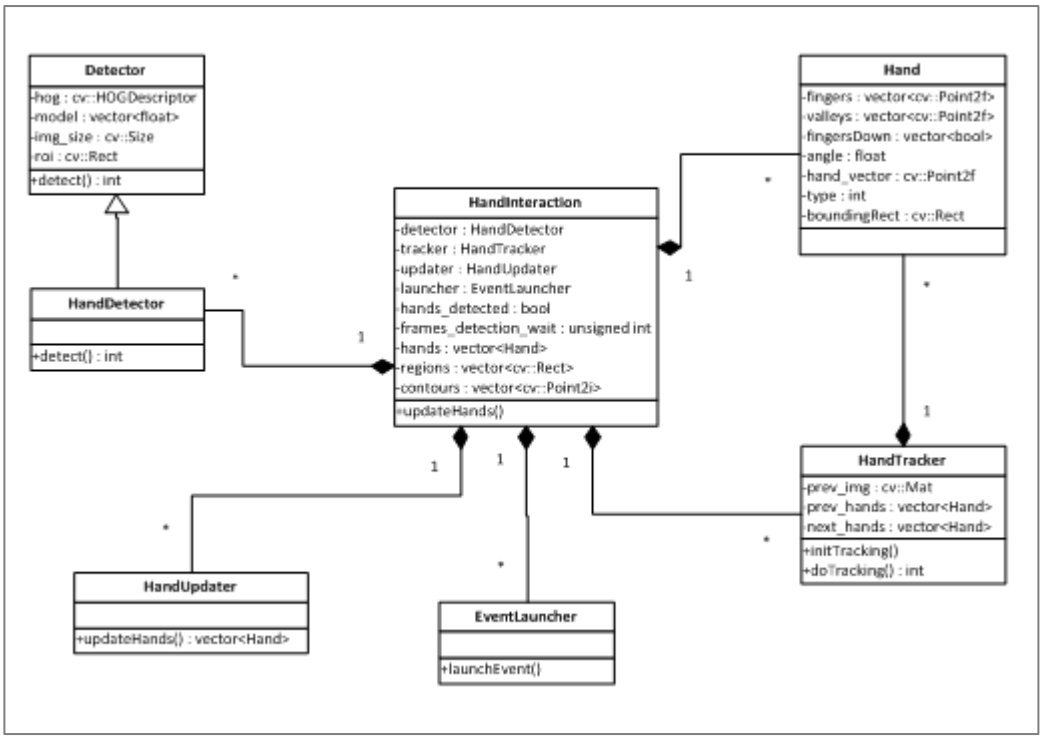

Ilustración 7.1 - Diagrama de clases de la implementación realizada

La clase *HandUpdater* se encarga de realizar parte de las operaciones de la etapa de seguimiento, más concretamente la fase de actualización de las características de la mano.

La clase *HandTracker* se encarga de implementar las operaciones de preparación y seguimiento siguiendo el método de *Lucas-Kanade*, y obtener la posición de los puntos calculados en el frame anterior.

La clase *EventLauncer* se encarga de realizar la interpretación de las características de la mano y lanzar eventos de teclado o ratón en función de los gestos que ésta este realizando en un determinado momento.

8. Conclusiones y trabajos futuros

La idea concebida en un primer momento de realizar un sistema, lo más genérico posible y en cualquier ambiente, de interacción entre usuario y computador, se ha visto finalmente limitada por las numerosas dificultades que se han encontrado durante su implementación.

La primera dificultad surgió con el uso de la API de OpenCV, que aun siendo una librería muy potente en el ámbito de la visión artificial, requiere una larga curva de aprendizaje. Esto supuso una larga fase de documentación[9][10] e investigación de las diferentes herramientas de esta API.

Otra dificultad surgió a la hora de elegir el método más estable y eficiente para realizar esta detección debido al gran número de alternativas: en función del color, diferencia con el fondo, diferencia de frames consecutivos, y, la finalmente usada, detección por aprendizaje supervisado.

Una tercera dificultad se ha encontrado a la hora de trabajar en distintos ambientes con condiciones muy cambiantes de luces y sombras, colores y objetos que aparecen en la escena, además del ruido introducido por la cámara al capturar objetos que realizan movimientos rápidos. Estos problemas se han conseguido mitigar mediante el uso de distintas técnicas de filtrado (ecualización de histogramas, normalizaciones, suavizado, truncamiento por valores de umbral), localización de una zona de interés sobre la que delimitar los cálculos e identificación de contornos basada en gradientes de intensidad. Aun así todavía queda abierta una vía de mejora en este aspecto.

Los problemas que no se han podido solventar nos ha obligado a introducir ciertas limitaciones en las condiciones de uso de la aplicación: fondo uniforme en las proximidades de la mano y niveles de luminosidad y oscuridad contenidos.

A pesar de que es posible la detección de varias manos al mismo tiempo, hemos restringido que su uso se realice únicamente a una mano tras comprobar que la interpretación de gestos con ambas manos no resultaba intuitiva para el usuario.

La forma en la que se ha concebido la representación de la mano permite una gran versatilidad a la hora de reconocer gestos. Puesto que se localizan la puntas de los dedos y los valles de la mano, es sencillo ampliar el conjunto de gestos que se reconocen basándose en sus relaciones.

Por último cabe destacar que la interacción de la aplicación con el sistema sobre el que se está ejecutando se realiza mediante eventos del mismo. A cada gesto se ha asociado un tipo de evento predeterminado para poder realizar una simulación.

A partir de la aplicación obtenida, se puede continuar el trabajo en dirección a ampliar y mejorar distintos aspectos de la misma.

Siguiendo el orden en el que se realiza el algoritmo, una primera mejora a realizar es un aumento en la fiabilidad de la detección, de forma que el número de falsos positivos y falsos negativos sea prácticamente nulo. Esto se conseguirá por la inclusión de más y mejores imágenes de muestra, tanto positivas como negativas, en la colección usada en el aprendizaje.

Con el objetivo de eliminar las restricciones impuestas en la utilización de la aplicación, como es la necesidad de un fondo uniforme en las proximidades de la mano, se pueden añadir otro tipo de filtrados, como puede ser por colores, para usar en combinación del filtrado por gradientes de intensidad en la fase del cálculo del contorno.

La ampliación del conjunto de gestos reconocibles y un sistema de que permita variar la configuración de la forma de mapear los gestos con los eventos. De este modo se podrán tener varios perfiles de gestos relacionados con determinadas acciones útiles para aplicaciones concretas.

La mejora del rendimiento del algoritmo también es otro punto clave de posibles trabajos futuros. Siendo difícil disminuir la complejidad de los algoritmos utilizados, existe la posibilidad de usar el módulo de la API de OpenCV de aceleración por GPU.

Bibliografía

[1] OpenCV (2012). *OpenCV Documentation*. [en línea] disponible en: http://opencv.itseez.com/index.html.

[2] Joachims, T. (2008). *SVM-Light Support Vector Machine*. [en línea] disponible en: http://svmlight.joachims.org.

[3] Dalal, N. y Triggs, B. (2005). *Histograms of Oriented Gradients for Human Detection*. Montbonnot, France.

[4] Abe, S. (2010). *Support Vector Machines for Pattern Classification*. London: Springer.

[5] Suzuki, S. y Abe, K. (1985). *Topological Strucutral Analysis of Digital Binary Images by Border Following*. Computer Vision, Graphics and Image Processing.

[6] Lucas, B.D. y Kanade, T. (1981) *An Iterative Image Registration Technique with an Application to Stereo Vision*. IJCAI.

[7] Bouquet, J.Y. (2000) *Pyramidal Implementation of the Lucas Kanade Feature Tracker*, Intel Corporation, Technical Report.

[8] D'Ademo, N. (2012) *qt-opencv-multithreaded - A simple multithreaded OpenCV example application using the Qt framework*. [en línea] disponible en: http://code.google.com/p/qt-opencv-multithreaded.

[9] Bradski, G. y Kaehler, A. (2008) *Learning OpenCV: Computer Vision with the OpenCV Library*. Sebastopol: O'Reilly Media.

[10] Laganière, R. (2011). *OpenCV 2 Computer Vision Application Programming Cookbook*. Birmingham: Packt Publishing.