**Antonio Raúl Guijarro Contreras Florentín Pérez González Adrián Epifanio Rodríguez Hernández Christian Torres González**

**Sistemas de interacción persona-computador,  Universidad de La Laguna**

**Reconocimiento de gestos**

****

**ÍNDICE**

1. Substracción de fondo ………………………….………………… 3
   1. My BG Subtractor Color ……………………..…………..… 3
   2. Ventajas e inconvenientes del Método ………...………… 3
2. Selección del contorno de la mano ..……………………………. 5
   1. Reducción de ruido ……………………...……………….… 5
   2. Generación del contorno de la mano ……..……………… 6
3. Convex Hull y Convexity Defects ……………………………..… 7
4. Bounding Rect …………………………………………………..… 8
5. Reconocimiento de gestos …………………………….…… 9
6. Pintar en la pantalla con el dedo ………………………… 10
7. **Substracción del fondo**

**1.1 My BG Subtractor Color**

La sustracción de fondo la realizamos en la clase "MyBGSubtractorColor". Para ello creamos un objeto de la clase MyBGSubtractorColor al que le pasamos la captura de la cámara. Lo primero que necesitamos es convertir los frames del video a HLS. Para ello creamos dos vectores para almacenar las muestras y un vector de medias donde almacenamos las medias y para ello hacemos uso de la función **createTrackbar()** para generar las barras de seguimiento de HLS.

A continuación en la función **LearnModel()** es donde convertimos la imagen a HLS. Para ello creamos un vector de muestras en el que almacenamos las posiciones de las esquinas de los cuadrados verdes que se muestran en la pantalla al elegir el frame de referencia. Con este frame se pretende eliminar todo el fondo mediante la comparación con los colores en HLS. El programa pide que cubras los cuadrados verdes que usará a modo de referencia para la substracción de fondo con la mano y pulses la barra espaciadora. Una vez hecho eso almacenará ese frame y lo convertirá a HLS mediante la función **cvtColor().**

**1.2 Ventajas e inconvenientes del Método**

Las ventajas de este método empiezan por la sencillez del mismo, es adaptable al medio, es decir, realmente no necesita que siempre tengamos que ejecutar el programa exactamente en el mismo sitio ya que calculamos las medias al principio de cada ejecución permitiéndonos así la posibilidad de que pueda ser usado por personas distintas e incluso luminosidades distintas.

Por otro lado, este método también tiene varias desventajas como es el hecho de que para calibrar el programa necesitamos que la mano esté a una distancia concreta de la cámara y durante la ejecución del mismo no se aleje demasiado porque dejaría de detectarla. Otra desventaja es el hecho de que para que solo detecte la mano, es necesario que haya un cambio de color de la mano al antebrazo siendo necesario taparlo para no "confundir" al programa ya que se basa en la detección de un rango de colores detectados de las medias recogidas al principio. A pesar de que el programa pueda ser ejecutado bajo distintas limosidades estas pueden afectar si varían durante la ejecución, es decir, el programa podría fallar si durante la ejecución del mismo varía mucho la luz haciendo así que la cámara captase la mano de otro color ligeramente distinto y que se saliese del rango predefinido.

**2. Selección del contorno de la mano**

En nuestro caso, lo primero que hacemos es crear la máscara binaria, la generaremos mediante la suma de los 18 frames con los cuadrados verdes anteriormente analizados y los cálculos con las medias almacenadas.

Para ello hacemos uso de la función **inRange()** cuya función es generar una imagen binaria a través de una imagen HLS y un escalar. Después sumamos los 18 frames generando así la imagen binaria final.

Ahora, para poder seleccionar el contorno de la mano necesitamos reducir el ruido de la imagen, este efecto se produce debido a la variación aleatoria del brillo o el color entre las imágenes o frames que se producen por el dispositivo de entrada.

**2.1 Reducción de ruido**

Comenzamos la reducción de ruido estableciendo los límites de los parámetros de la imagen mediante la función **threshold()** y luego aplicamos **medianBlur()** para suavizar los frames. Después, aplicamos dilatación seguida de erosión para intentar eliminar la mayor cantidad de ruido posible y terminamos generando la estructura de elementos con **getStructuringElement().**

* **Dilatación**: consiste en aumentar el tamaño de los puntos blancos que forman la imagen, de esta manera si existe algún punto negro dentro de lo que sería la mano, al dilatar, este desaparecería ya que estamos aumentando el tamaño de los puntos blancos.
* **Erosión**: consiste en disminuir el tamaño de los puntos blancos de la imagen, de esta manera si existen zonas blancas que no forman parte de la mano, al aplicar erosión acabarían desapareciendo. El único inconveniente de esta segunda opción es que el tamaño de la mano quedaría reducido ya que estamos haciendo más pequeños la parte blanca.

**2.2 Generación del contorno de la mano**

Para generar el contorno de la mano fue necesario que antes pintáramos un pequeño círculo en la pantalla ya que de lo contrario si el programa no encuentra un contorno fallaría el programa, es decir, siempre debe ser capaz de reconocer un contorno y de esta manera estamos "forzando" el que reconozca al menos el pequeño círculo. En caso de que en el frame aparezca un contorno reconocible mayor el programa pasa a generar el contorno del nuevo en lugar del círculo.

Mediante la función **findContours()** el programa es capaz de generar el contorno de la mano que luego imprimiremos en la pantalla.

**3. Convex Hull y Convexity Defects**

Una vez generado el contorno le añadimos los puntos de convexidad, son los puntos donde la función de la línea del contorno pasa de ser decreciente a creciente o viceversa. Estos puntos sirven para indicarnos y mostrarnos en la pantalla donde están localizadas las puntas de los dedos y la unión entre ellos.

Para calcular estos puntos de convexidad, que nosotros hemos separado entre los puntos de dedo (puntos rojos), y los puntos de unión (puntos verdes), hemos aplicado formulas relacionadas con los ángulos entre dedos y la longitud entre el vértice del ángulo y los extremos de los dedos de la mano. De este modo somos capaces de imprimir correctamente los puntos en el sitio correspondiente. Por ejemplo, para calcular los puntos verdes comprobamos que el ángulo sea menor de 90º y la longitud de sus extremos menor que un error introducido por defecto en el programa.

Además, hemos implementado un contador para cada uno de los puntos para saber la cantidad de dedos que hay mostrados a la cámara en cada momento. Como los puntos verdes (puntos de unión) son más precisos decidimos usar esos cuando fuera posible a excepción de cuando solo hubiese un dedo levantado o ninguno en cuyo caso no es posible ya que el programa no detectaría ningún punto verde. En dicho caso, imprimimos la cantidad de puntos rojos que está detectado, es decir, si detecta uno es que se está mostrando un dedo y en caso de que no detecte ninguno se detectaría que la mano está cerrada.

**4. Bounding Rect**

Para una mejor implementación del programa y mayor modularidad del mismo hemos optado por usar la función **boundingRect()** para generar el rectángulo mínimo posible del contorno de la mano. Con este rectángulo sacamos proporciones que luego aplicamos a todo el programa.

Un ejemplo de esto es el caso del porcentaje de error que empleamos para la longitud de los dedos. Si pusiéramos un tamaño fijo en píxeles como podría ser por ejemplo 50 píxeles esto no sería preciso ya que no equivale a la misma longitud de tu mano 50 píxeles cuando la mano está a 10 centímetros de la pantalla a cuando está a 2 metros.

También mediante esta función hemos calculado así el punto medio del rectángulo que la mayoría de las veces coincide con el de la mano y gracias a este punto podemos hacer más sencillas o desarrollar varias funciones calculando la posición de dicho punto.

**5. Reconocimiento de gestos**

Mediante el contador de los dedos, somos capaces de detectar cuantos dedos hay en la pantalla en cada momento, además hemos implementado una función que detecta el movimiento de la mano. Esta función se encarga de imprimir en la pantalla hacia donde se está moviendo, se acaba de mover la mano o si esta parada. Para depurar esta función hemos empleado el punto medio del rectángulo generado por el **boundingRect()**. Cada cierto tiempo el programa guarda la posición del punto medio del cuadrado como posición inicial de la mano, pasados "x" milisegundos vuelve a comprobar la nueva posición y en función de esa nueva posición es capaz de determinar el movimiento realizado.

También hemos intentado desarrollar la funcionalidad de saber el ángulo en el que está girada la mano hacia la derecha o hacia la izquierda. En esta parte decidimos calcular la media de los ángulos entre los puntos rojos y verdes y los puntos medios de la base y la parte alta del **boundingRect()** que generamos anteriormente alrededor de la mano. Esto no ha generado varias dificultades a la hora de imprimir los ángulos asique para probar decidimos imprimir por pantalla el ángulo que se estaba mostrando en cada momento, ángulo que aparentemente no tenía sentido alguno. Como la impresión el ángulo no nos ayudaba decidimos imprimir también la línea que coge como referencia sobre la que se calcula el ángulo, es decir, la línea entre el punto medio de la base y la parte alta del **boundingRect().** Esta línea nos hizo ver que el ángulo se estaba calculando de manera errónea ya que no estaba cogiendo los **tl()** y **br()** del **boundingRect()** sino de la pantalla completa.

**6. Pintar en la pantalla con el dedo**

Por último, desarrollamos la funcionalidad de poder pintar en la pantalla mediante el movimiento de un dedo. Esto lo implementamos mediante el almacenamiento de todos los puntos por los que pasa el dedo en un vector que se va pintando sobre la imagen.

Para activar esta funcionalidad hay que cerrar el puño y una vez se levante un dedo después de eso el programa pasa a pintar en la pantalla todos los puntos por los que pase el extremo del dedo levantado. Para borrar lo pintado y parar de pintar basta con volver a cerrar el puño de la mano.