**1º. Sustracción de fondo**

**1.1 Conversión a HLS y obtención de medias.**

En este primer apartado se explicará de manera fácil y sencilla la conversión de la imagen capturada por la cámara(frame) a una imagen en formato HLS y como el programa sabrá diferenciar la mano.

En primer lugar, debemos convertir el frame obtenido por la cámara, (código ya implementado) de RGB a HLS ya que nosotros queremos trabajar con los valores de Hue (Matiz), Ligthness (Brillo) y Saturation (Saturación) en vez de Red, Green y Blue. Esto se obtiene haciendo uso de la función cvtColor.

Una vez convertido el frame a HLS, ya que nuestro objetivo es reconocer gestos con la mano, deberemos indicar al programa que es lo que hay que considerar parte de la mano y que es lo que no. Para ello se utilizan los cuadrados verdes que aparecen a la hora que capturar la foto de la mano. La función de dichos cuadrados es la de obtener el valor medio de los colores que se encuentran en su interior, para que a la hora de analizar cada frame obtenido por la cámara, el programa sepa detectar que es lo que forma parte de la mano. Dicha media será almacenada en un vector de medias, el cual se utilizará para comprobar si el color contenido en un pixel del frame es parte de ese vector.

El paso 1.1, se encuentra implementado en el fichero “MyBGSubtractorColor.Cpp” y dentro de este fichero, en la función MyBGSubtractorColor.Cpp::LearnModel()

**1.2 Creación y obtención de imagen binaria**

En este segundo apartado se explicará de manera fácil y sencilla la obtención de la imagen binaria del frame capturado por la cámara y como se representa dicha imagen.

Una vez hemos obtenido las medias de los colores de la mano en el apartado anterior, en este segundo, empezaremos estableciendo unos límites de colores con los que podremos decir que pixeles de la imagen forman parte de la mano. Para ello utilizamos los valores del vector de medias a los cuales le restaremos los valores, los cuales podremos regular a nuestra medida, para indicar si ese pixel forma parte de la mano. En caso de que el pixel forme parte, dado que estamos representando la imagen binaria, vamos a establecer el valor a 255 para que lo pinte de blanco, en caso contrario, es decir, en caso de que el pixel no forme parte de la mano, indicaremos que su valor es de 0 para que lo pinte de negro. Como ya se ha comentado anteriormente esto es gracias a los parámetros con los que trabaja HLS, los cuales podremos ir modificando.

Una vez establecido dicho limite, mediante la función inRange () detectaremos que pixeles están dentro del rango establecido para que se considere mano, y se guardara dicha imagen en un vector. Una vez realizado este proceso con la imagen se procederá a formar la imagen binaria sumando todas estas imágenes con la imagen “background” ya creada anteriormente, la cual se había inicializado toda a negra. Al finalizar la suma de todas estas imágenes, solo quedaría copiar dicha imagen binaria en el atributo que ya habíamos creado para esto, ya que “background” era una variable local utilizada de manera permanente.

El paso 1.2, se encuentra implementado en el fichero “MyBGSubtractorColor.Cpp” y dentro de este fichero, en la función MyBGSubtractorColor.Cpp::ObtainBGMask()

* Ventajas y desventajas de los métodos empleados en 1.1 y 1.2

Las ventajas de este método empiezan por la sencillez de este, 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 de este 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 de este varía mucho la luz haciendo así que la cámara captase la mano de otro color ligeramente distinto y que se saliese del rango predefinido.

**2.1 Reducción de ruido**

En este tercer apartado se explicará de manera fácil y sencilla el procedimiento empleado para reducir el ruido en la imagen binaria, es decir obtenerla lo mas limpia posible de tal manera que se asemeje lo máximo posible a la mano real.

Previo a explicar la implementación de código debemos hacer una breve explicación de cómo funciona la reducción de ruido. Para ello existen dos técnicas:

* **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 seria 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 mas pequeños la parte blanca.

Una vez hecha esta introducción,