Informe SIPC: Reconocimiento de gestos

Adán de la Rosa Lugo
Juan Siverio Rojas
Alberto Rodriguez Fuentes
Pablo Torres Albertos

INTRODUCCIÓN

En esta práctica vamos a aprender a realizar un programa de reconocimiento de gestos de la mano. Hemos agregado algunas funciones extra al programa, como una interfaz gráfica hecha en QT y agregando barras deslizantes para los filtros de eliminación de ruido como la mediana , dilatación y erosión. Nuestro programa, es capaz de reconocer dos gestos simples, (pistola y saludo), contar el numero de dedos y pintar/borrar en la pantalla.

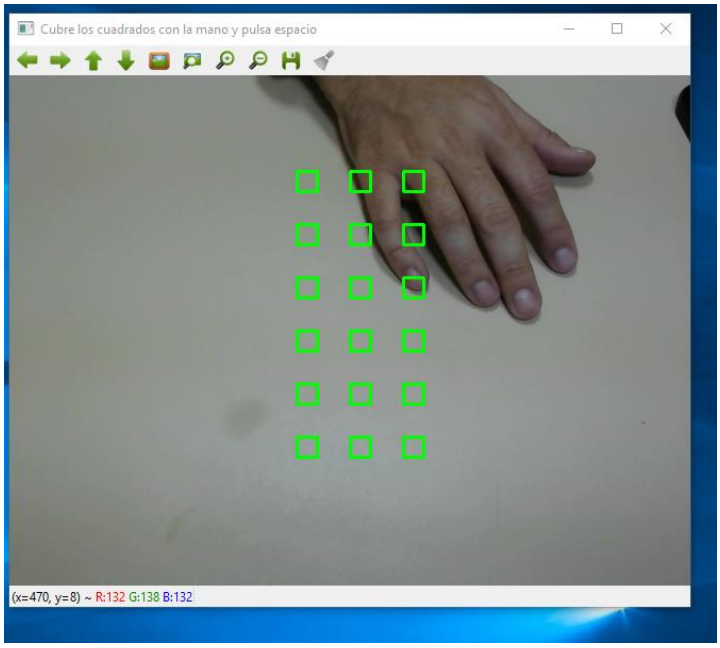
SUBSTRACCIÓN DE FONDO

En primer lugar, deberemos procesar la imagen para quedarnos solo con el contorno de la mano, eliminando todo lo demás. Esto lo haremos diferenciando la mano, la cual tendrá un color blanco y el resto, que no nos interesa, en negro. Para ello, usaremos unas pequeñas regiones de la imagen representadas por cuadrados verdes donde se colocará la mano, las cuales llamaremos **regiones de interés. Hemos agregado sliders para poder aumentar/disminuir el número de regiones de interés a capturar, así como el tamaño y la distancia de cada uno de ellos.**

Una vez el usuario coloque la mano abarcando todos los cuadrados verdes y presione espacio, se habrán almacenado las diferentes regiones de interés y se habrán calculado las medias de cada una de ellas.

En código, este proceso se hace llamando a la función ***LearnModel()*** de la clase ***MyBGSubstractorColor*** la cual se encarga de dibujar las regiones de interés en la pantalla y :

- Pasa la imagen a hls
- Calcula la media de cada región de interés y lo almacena en el vector means.



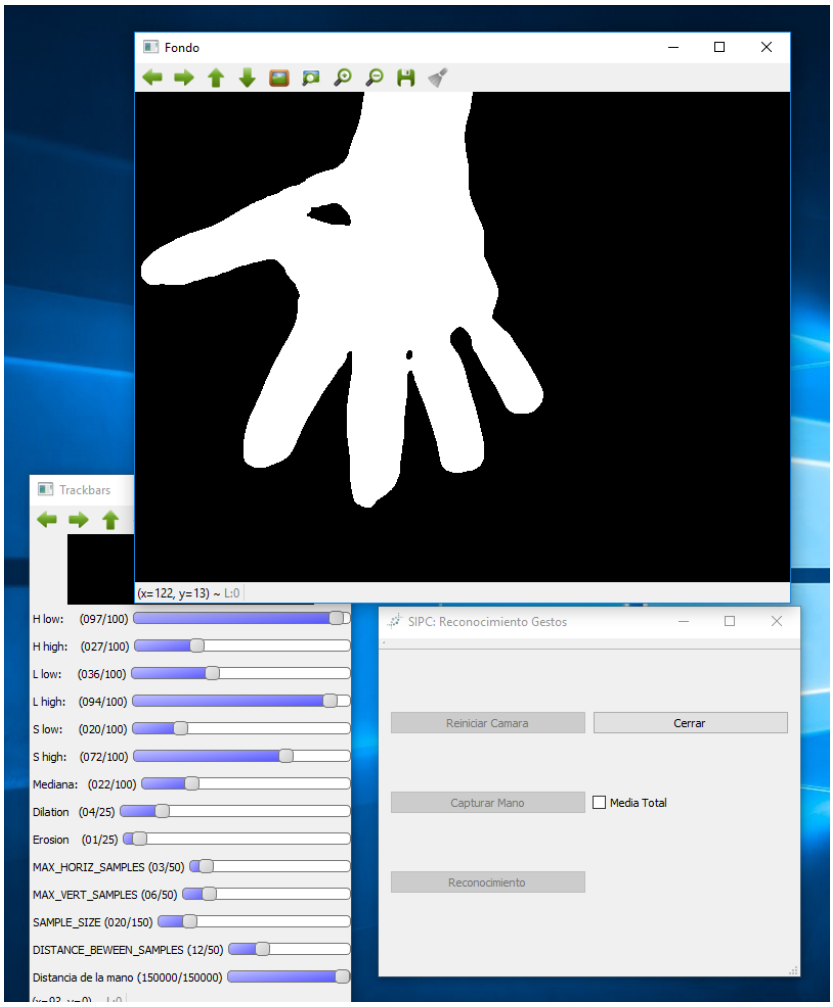
A continuación, obtendremos para cada frame, la máscara de fondo con las medias obtenidas. Esto lo haremos con la función

ObtainBGMask(frame,&bgmask) la cual, para cada frame define los rangos máximos y mínimos para cada canal hls, y si el punto de la imagen esta dentro del rango de los atributos (hue, lightness, saturation) se considerará como punto blanco y si se sale de este, como punto negro.

Nota: a la hora de implementar la definición de rangos hay que tener en cuenta que se pueda salir del rango [0-255] las operaciones de resta o suma.

Una vez definidos los rangos, la función **inRange(hls_frame, low, high, tmp_frame)** nos obtendrá la **imagen binaria**. Una vez hecho esto, **acumulamos** las diferentes imágenes binarias para obtener nuestra **máscara de fondo.**

- **La ventaja** de utilizar este método es que nos permite establecer diferentes rangos con cada uno de los parámetros hls (utilizando las barras del programa) aportando mayor flexibilidad, pues podremos ejecutar nuestro programa en diferentes lugares y poder ajustar los parámetros a mano.
- **La principal desventaja** de este procedimiento es que todos los puntos del fondo que entren dentro del rango serán cogidos como puntos de la mano por lo que se generará **ruido** en la imagen.



ELIMINACIÓN DE RUIDO

Hemos visto en clase diferentes técnicas para solucionar el ruido generado en la imagen como son el **filtro de la mediana** u operaciones morfológicas de **dilatación** y **erosión**.

- **La mediana:** escoge el valor central de un conjunto de píxeles, quedándose solo con ese valor y ese pixel de entre todo el conjunto. Esto reduce la resolución de la imagen y funciona mejor cuando hay pequeños ruidos granulares.
- **dilatación**: dilata los pixeles en blanco mediante un elemento estructurante de longitud impar, normalmente con forma circular o cuadrada, el cual actúa sobre todos los puntos blancos, convirtiendo en blanco todos los pixeles vecinos. Los trazos blancos se hacen más gruesos.
- **erosión:** erosiona los pixeles en blanco, funciona igual que para la dilatación pero teniendo en cuenta que aquí los pixeles vecinos se hacen negros y no blancos, obteniendo trazas blancas más finas.

En nuestro código, hemos implementado las tres y las hemos asociado a barras deslizantes para que sea el usuario quien ajuste los parámetros a su gusto. En particular, hemos detectado una mejora en la imagen cuando hemos aplicado primero la dilatación y luego la erosión.

DETECCIÓN DE CONTORNO DE LA MANO

Para detectar el contorno de la mano, usaremos la función de opencv ***findCountours()*** la cual almacenará en la variable **contours** todos los contornos en blanco de la imagen. Como solo queremos dibujar el contorno de la mano, tendremos que **quedarnos con el vector de puntos de mayor tamaño.** Por último, pintamos en la imagen de salida el contorno con la función ***drawContours()***.

Hay que tener en cuenta que, si la imagen no contiene ningún punto en blanco, el programa terminará con un error, para solucionarlo, dibujamos un pequeño círculo blanco en una de las esquinas.

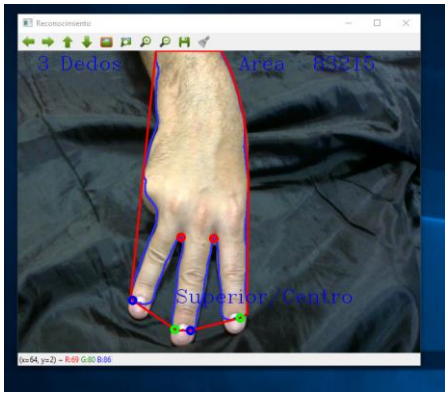
Generar la malla convexa

Una vez detectado el contorno, procedemos a obtener la malla convexa con la función ***convexHull().*** Esta malla unirá las puntas de los dedos con la muñeca de las manos

Defectos de convexidad

Son los puntos del contorno que se separan de la malla convexa, es decir, las zonas que hay entre los dedos de nuestra mano y están delimitadas por tres puntos: start, end y fardest y, además, por una profundidad y por un ángulo.

Para filtrar todos los defectos de convexidad, nos hemos quedado con los puntos de mayor distancia (fardsest), y por aquellos que tienen un Angulo menor de 100 grados entre los tres puntos.



Calcular rectángulo mínimo
Por último, mediante el uso de la función ***boundingRect()*** calcularemos el rectángulo mínimo de la mano. Esto lo hacemos para poder diferenciar al contar los dedos uno y el cero, ya que el problema que se nos presentaba es que con un solo dedo levantado no tenemos defecto de convexidad. Para diferenciar estos dos casos, calculamos el área del rectángulo con la mano abierta y si esta área disminuye más de un 50% estará la mano cerrada y si no, será un solo dedo.
Esta solución que hemos implementado tiene un inconveniente importante, y es que, si el usuario aleja o acerca la mano, el cálculo será erróneo, pero se podrá ajustar mediante las barras deslizantes para calibrarlo.

DETECCION DE GESTOS

Para detectar los gestos de la mano lo que hicimos fue detectar gestos sencillos como por ejemplo la forma de pistola, lo que hicimos fue combinar el bounding Rec comprobando que si el ancho era mayor que el alto y que había un solo punto de convexidad, entonces detectaba que la mano tenia forma de pistola.

También detecta la ubicación de la mano diferenciadas en seis estados: Superior/Izquierda, Superior/Derecho, Inferior/Izquierda, Inferior/Derecha, Centro/Superior y Centro/Inferior, todo ello mostrando en pantalla en todo momento.

PINTAR EN LA PANTALLA

Para pintar en la pantalla habíamos tenido en cuenta que si el usuario coloca la mano haciendo el símbolo de la victoria, podría pintar con el punto de convexidad mas profundo de la mano (el que esta entre los dos dedos). Si se cierra la mano, se borrara todo lo pintado en pantalla.