**Visión por Computador**

17

**Trabajo 3**

**CONTORNOS**

**Jorge Andrés - 679155**

**Javier Aranda - 679184**

**INTRODUCCIÓN**

El objetivo de este este tercer trabajo es desarrollar mediante “OpenCV” un programa que detecte los contornos de una imagen y sea capaz de, a partir de ellos, calcular el punto de fuga de un pasillo. Para el cálculo de los contornos se han estudiado los operadores de “Sobel”, “Scharr” o “Canny” con un posible filtro Gaussiano previo para reducir el ruido de la imagen.

// SEGUIR

**LECTURA DE IMAGEN**

Para la parte obligatoria del trabajo, se ha desarrollado un programa que leyera una cierta imagen de la ruta indicada como parámetro. Estas imágenes, para indicar el modo de lectura al método “imread”, se comprobaron si estaban en blanco y negro o a color. Aunque pareciera algo que no podía tener relevancia, era necesario por si se debía aplicar una conversión de RGB a escala de grises, ya que el mejor método para la búsqueda de contornos es utilizar una imagen en escala de grises.

Una vez se leía la imagen, se comprobaba si se había conseguido con éxito o, por el contrario, si no existía y había ocurrido un error, se abortaba el programa indicándolo. Si se había podido leer la imagen correctamente, se mostraba por pantalla en una ventana para poder ver sobre ella todo el procedimiento que se le fuera aplicando hasta la detección final del punto de fuga.

// IMAGEN LEÍDA

**CÁLCULO DEL GRADIENTE**

Tras tener ya la imagen indicada en escala de grises, el siguiente paso era la detección de los puntos de contorno. Para ello, se aplicó en primer lugar un filtro Gaussiano con sigma variable, ya que las máscaras que se iban a utilizar para el cálculo del gradiente de la imagen eran muy sensibles al ruido. Con ello conseguíamos una imagen filtrada que ofrecería mejores resultados finales que aplicando la detección del punto de fuga a la imagen original directamente.

Para aplicar el filtro Gaussiano se utilizó la función “GaussianBlur” de “OpenCV”. Dicha función recibía como parámetros la imagen original, así como una matriz para guardar la imagen con el filtro aplicado. Además, se le podía indicar el tamaño del “kernel”, y en caso de ser 0, la desviación típica en cada eje que sería la que tuviera en cuenta. Con este método ya teníamos la imagen lista para seguir el procedimiento.

// IMAGEN FILTRO GAUSSIANO Y RESULTADO

A partir de la imagen filtrada, se estudiaron los posibles operadores capaces de calcular el gradiente en cada uno de los ejes. Para ello, se miró la documentación de los operadores de “Sobel” y “Scharr”, que eran máscaras que se aplicaban a toda la imagen y devolvían la aproximación del gradiente de la función de intensidad de una imagen. La diferencia principal de los dos métodos es que “Scharr” aproximaba mejor la primera derivada que la función de “Sobel”.

// IMAGEN KERNEL Y SOBEL

En primer lugar, se utilizó el operador de “Sobel” que recibía como parámetros la imagen filtrada, así como una matriz para devolver el gradiente indicado. Dicho gradiente que nos interesaba era en dirección del eje “X” e “Y” para poder calcular a partir de ellos el ángulo u orientación hacia la que iba la función de iluminación indicada. Para distinguirlos, se realizó la llamada a esa función con los parámetros “0” y “1” en los dos ejes según se quisiera el eje “X” o “Y”. La única particularidad de esta función es que pedía el tipo de dato en el que debía devolver los valores del gradiente y, en nuestro caso, usamos el tipo “CV\_32F” ya que era uno de los que aportaba más precisión.

Una vez calculados los gradientes, se probaron a mostrar en dos ventanas para comprobar que el procedimiento se estaba siguiendo satisfactoriamente. Para ello, ya que el rango para mostrar una imagen en escala de grises era [0,255], y el gradiente obtenido estaba en el rango [-255,255] según fuera de claro a más oscuro o viceversa, se aplicó una transformación dividiendo el valor por de cada miembro de la matriz por el escalar 2 y sumándole 255. Con estos sencillos cálculos se conseguía que los valores entraran en el rango esperado.

Al mostrar la imagen por pantalla, nos encontramos con que los resultados no eran los esperados, apareciendo cosas que no tenían ningún sentido. Al repasar varias veces el código que llevábamos hasta el momento, no observábamos ningún fallo que pudiera provocar esos resultados, así que miramos la documentación del método “imshow” de “OpenCV” para intentar localizar el fallo. Al observarla detenidamente nos dimos cuentas que realizaba un procedimiento u otro según fuera el tipo de la matriz que se le pasaba como parámetro.

Si la matriz que recibía era de datos de tipo entero, no realizaba ninguna conversión y los mostraba tal cual, pero si éstos eran de tipo “float”, ya fuera de 32 o 64, lo que hacía era multiplicar los valores por 255 para introducirlos en el rango [0,255]. Al ver los valores que estaban contenidos en nuestra matriz ya estaban en ese rango, nos dimos cuenta que los estaba multiplicando por 255 sin que hubiera necesidad, así que decidimos dividirlos por el escalar 255 para tenerlos en el rango [0,1] y que al dibujarlos lo hiciera correctamente.

Tras conseguir mostrarlos ya como se esperaba, se vio que el gradiente en el eje “X” era como el de las imágenes de prueba proporcionadas, pero, sin embargo, en el eje “Y” los colores estaban cambiados; es decir, donde debía ir blanco era negro y viceversa. Tal y como estaban definidas las funciones de gradiente, el vector gradiente es normal a la superficie apuntando al lado claro, por lo que nuestro eje “Y” estaba invertido a como lo queríamos tener. Para solucionarlo, simplemente se multiplicó la matriz por el valor “-1” y ya estaba según como lo deseábamos.

// IMAGEN DE GRADIENTE EN X e Y

Con los gradientes ya calculados y comprobado que eran correctos, se pasó a calcular tanto su orientación como su módulo. Para ello, teníamos definidas las siguientes fórmulas que nos permitían obtener ambas magnitudes de una manera sencilla:

// FÓRMULA ORIENTACIÓN Y MóDULO

Para calcularlo, se estudió en primer lugar recorrer secuencialmente la matriz de gradiente en eje “X” e “Y” e ir obteniendo los valores, insertándolos en último término en otra matriz que albergaría todos los resultados. Sin embargo, estudiamos la documentación de “OpenCV” por si existía alguna función que ya hiciera lo que esperábamos, viendo que efectivamente sí que estaba ya disponible y era sencilla de usar.

La función se llamaba “cartToPolar” y recibía como parámetros las matrices de los gradientes en ambos ejes, así como las que devolvería los resultados de calcular el módulo y la orientación. Además, aceptaba como parámetro extra si se quería la orientación en grados o radianes, estando por defecto radianes. La única duda que quedaba es en el rango que devolvía el ángulo, ya que lo deseábamos para trabajar con mayor facilidad y mostrarlo en el rango [0,2\*PI]. Al mirarlo detenidamente, se vio que lo devolvía justo en ese rango, así que se utilizó y se obtuvieron los resultados.

Para comprobar si eran correctos o no, se realizó el mismo procedimiento que con el gradiente, se probó a mostrarlo en una ventana nueva. El módulo estaba ya en el rango de [0,255] que se debía mostrar en una imagen, simplemente se aplicó la división para dejarlo entre 0 y 1 al traterse de datos de tipo “float” como se explicó anteriormente. Para el ángulo, era necesario pasarlo a [0,255] para ver todo con más exactitud. Así pues, se dividió entre PI y se multiplicó por 128, consiguiendo el resultado esperado. Finalmente se dividió entre 255 para dejarlo en el rango aceptado para datos “float”.

// IMAGEN ORIENTACIÓN Y MÓDULO.

Al mostrarlos por pantalla, se pudo ver que claramente eran muy parecidos a los proporcionados para realizar las comprobaciones de cada paso del procedimiento, y ver que se seguía correctamente. El único punto distinto a destacar era que el módulo salía con tonos muchos más claros que el de las transparencias, algo debido a que se había dividido por algún escalar antes de mostrarlo, por lo que no era ningún problema grave. En lo que al ángulo se refiere, se apreciaba que estaba perfecto en los contornos, aunque como aspecto a destacar eran los contornos verticales. Esos puntos adquirían color blanco o negro indistintamente, algo que pudiera parecer que estaba mal, pero que era correcto ya que el ángulo podía coger valores de [0,10] o [350,360] haciendo cambiar radicalmente el color.

**PUNTOS DE CONTORNO**

Tras haber realizado los pasos anteriores exitosamente,