Trabajo 2. Contornos

Visión por Computador

Alejandro Paricio García, 761783
Fernando Peña Bes, 756012

Universidad de Zaragoza
21 de marzo de 2021

Índice

1. Introducción

En la segunda práctica de la asignatura se plantea el problema de detección de contornos aplicado a la obtención del punto de fuga de un pasillo. Para ello, se ha seguido la estructura sugerida por los apartados de la práctica. Primero, se ha implementado el cálculo de los gradientes, su módulo y su orientación mediante el operador de Canny para suplir su ausencia en las funciones de OpenCV, y así poderlo comparar con otros operadores. Acto seguido, se emplea la transformada de Hough para obtener rectas en la imagen que, gracias a un postprocesado, van a permitir la obtención del punto de fuga en las imágenes proporcionadas, además de en un vídeo de un pasillo grabado con este propósito. A continuación, se comentan las decisiones tomadas a lo largo del transcurso de la práctica, así como su motivación y sus consecuencias.

2. Detección de contornos

El primer paso es la detección de contornos es el cálculo de los gradientes de la imagen. Para ello, se han probado tres operadores: Sobel, Scharr y Canny.

Para cada uno de ellos se muestra el gradiente en X y en Y, la magnitud, la orientación y el contorno detectado a partir del gradiente.

En todos los casos, ambos gradientes se han normalizado entre -255 y 255 de forma conjunta manteniendo la proporción entre ambos. Para calcular la magnitud y la orientación se ha utilizado la función `cartToPolar` de OpenCV, a la que se le pasan las matrices de los gradientes en X y en Y, y devuelve dos matrices con el ángulo y la magnitud de los vectores que forman las matrices de los gradientes.

La detección de los contornos se realiza con la función sobrecargada `Canny` de OpenCV, que toma como entrada las dos matrices con los los gradientes en X y en Y de la imagen y dos umbrales de histéresis. Esta función encuentra los contornos buscando primero los máximos locales del gradiente y aplicando a continuación el algoritmo de `Edge linking` con histéresis.

En todos los ejemplos se ha usado `treshold1`=20 y `treshold2`=40 como umbrales de histéresis.

Siguiendo la convención de las direcciones de los ejes de OpenCV, hemos considerado que el eje X va hacia la derecha de la imagen, y el Y hacia abajo.

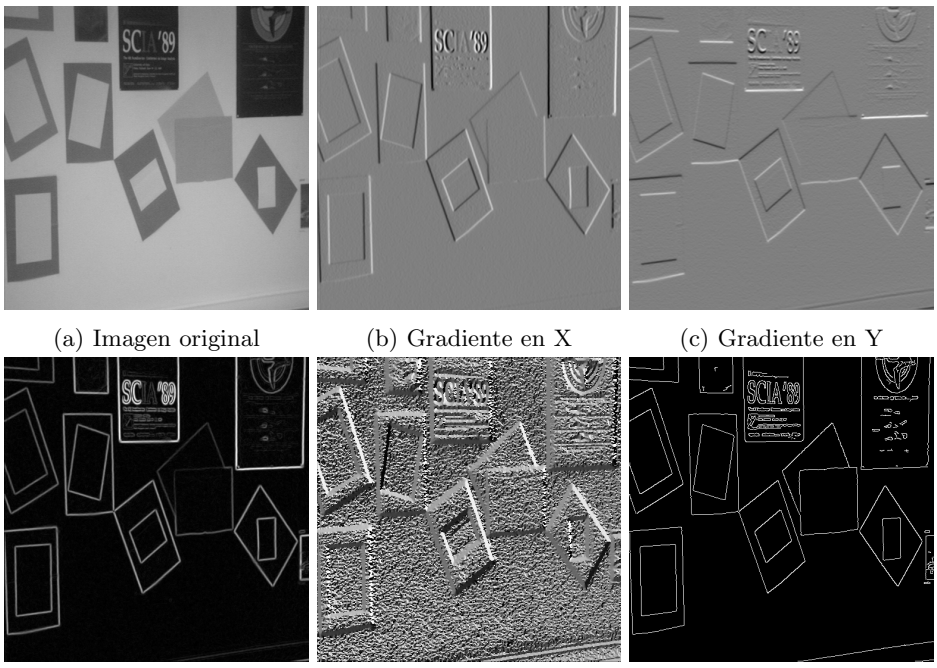
2.1. Operador de Sobel

Para aplicar el filtro de Sobel se ha usado la función `Sobel` de OpenCV. Previamente se ha aplicado un filtrado Gaussiano mediante un kernel de tamaño 3 y desviación típica 1, con el fin de reducir el impacto del ruido de la imagen en el gradiente.

Los gradientes se calculan utilizando las siguientes máscaras de convolución:

$$\nabla_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I, \quad \nabla_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I \tag{1}$$

En la Figura 1 se muestran los resultados al aplicar este filtro.



(a) Imagen original (b) Gradiente en X (c) Gradiente en Y

(d) Magnitud (e) Orientación (f) Contorno

Figura 1: Gradiente calculado usando el operador de Sobel

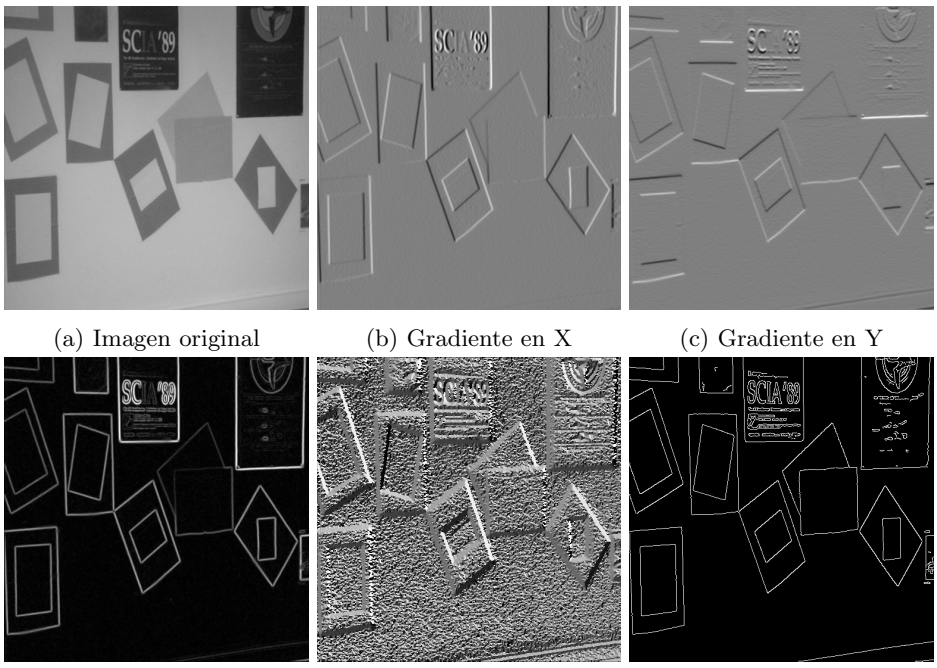
2.2. Operador de Scharr

El filtro de Scharr es una versión optimizada del de Sobel que intenta alcanzar la simetría rotacional perfecta. Se ha usado la función `Scharr` de OpenCV, y también se aplica como paso previo el mismo filtro Gaussiano que en el apartado anterior.

Los kernels aplicados por la función son los siguientes:

$$\nabla_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix} * I, \quad \nabla_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix} * I \tag{2}$$

En la Figura 2 se muestran los resultados al utilizar el filtro de Scharr.



(a) Imagen original (b) Gradiente en X (c) Gradiente en Y

(d) Magnitud (e) Orientación (f) Contorno

Figura 2: Resultados de aplicar el operador de Scharr

2.3. Operador de Canny

El operador de Canny, como se ha comentado en las clases de teoría, está diseñado para se más robusto frente al ruido, la localización del contorno se realiza con mayor precisión y evita devolver contornos duplicados. Este filtro no se encuentra en OpenCV, por lo que se ha implementado a mano.

La máscara de convolución de este operador, al igual que la de los anteriores es separable. Está formada por un kernel Gaussiano y el kernel de la primera derivada de la Gaussiana.

Utilizando los polinomios de Hermite [1], el kernel Gaussiano en la dirección del eje X se puede calcular de la siguiente manera:

$$G_\sigma(x) = \exp\left(\frac{-x^2}{2\sigma^2}\right)$$

De forma similar también se puede calcular el kernel de la derivada de la Gaussiana.

$$G'_\sigma(x) = \frac{-x}{\sigma^2}\exp\left(\frac{-x^2}{2\sigma^2}\right)$$

El valor σ corresponde con la desviación estándar de la distribución gaussiana que siguen los kernels. El tamaño de la máscara suele escogerse como 5σ o 7σ. A continuación se muestra un ejemplo de los dos kernels para el eje X utilizando $\sigma = 1$ y tamaño $n = 5$:

$$G_1(\begin{bmatrix}-2 & -1 & 0 & 1 & 2\end{bmatrix}) = \begin{bmatrix}0{,}1353 & 0{,}6065 & 1 & 0{,}6065 & 0{,}1353\end{bmatrix}$$
$$G'_1(\begin{bmatrix}-2 & -1 & 0 & 1 & 2\end{bmatrix}) = \begin{bmatrix}0{,}2507 & 0{,}6065 & 0 & -0{,}6065 & -0{,}2707\end{bmatrix}$$

En cuando al eje Y, tanto para el filtro Gaussiano como para la derivada se cumple que

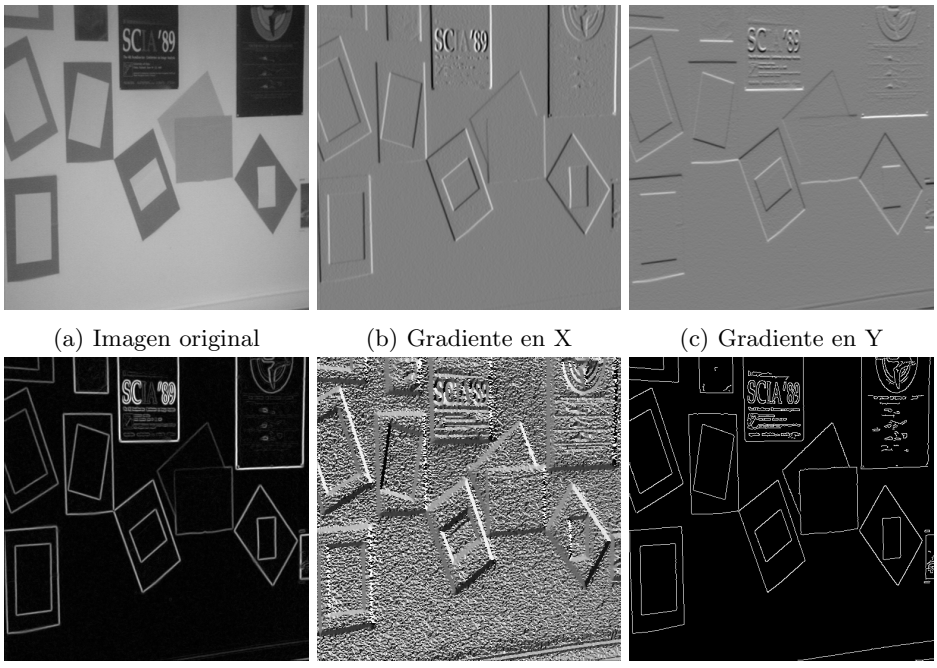
$$G_\sigma(y) = G_\sigma(x)^\top$$
$$G'_\sigma(y) = G'_\sigma(x)^\top$$

Una vez calculados los kernels, se pueden pueden aplicar de forma separable como se muestra a continuación:

$$\nabla_x = G_\sigma(x) * [-G'_\sigma(y) * I] \tag{3}$$
$$\nabla_y = -G'_\sigma(x) * [G_\sigma(y) * I] \tag{4}$$

Los signos − se han añadido para que el gradiente calculado respete la convención de las direcciones de los ejes que se ha propuesto para la práctica.

Los resultados al aplicar el operador de Canny se encuentran en la Figura 3.



(a) Imagen original (b) Gradiente en X (c) Gradiente en Y

(d) Magnitud (e) Orientación (f) Contorno

Figura 3: Gradiente calculado usando el operador de Canny

2.4. Operador escogido

En la imagen que se ha utilizado para comparar los operadores no se aprecian demasiadas diferencias entre unos y otros. A pesar de ello, por lo motivos de robustez que se han comentado, se ha decidido utilizar el operador de Canny para el siguiente apartado de la práctica. Este operador además realiza el filtrado Gaussiano y el cálculo de los gradientes de forma simultánea, por lo que más eficiente para procesar vídeo en tiempo real ya que no requiere un filtrado previo.

3. Obtención del punto de fuga

En el siguiente apartado de la práctica se plantea el problema de obtención del punto de fuga de un pasillo. Para ello, se pide emplear la transformada de Hough. En este caso se ha decidido emplear la función `HoughLines` de OpenCV, ya que cuenta con todas las características necesarias para la resolución del problema.

La anterior se basa en la expresión de líneas como coordenadas polares (ρ, θ). Para cada punto (x_0, y_0), se puede definir el conjunto de líneas que pasan por el mismo mediante la expresión:

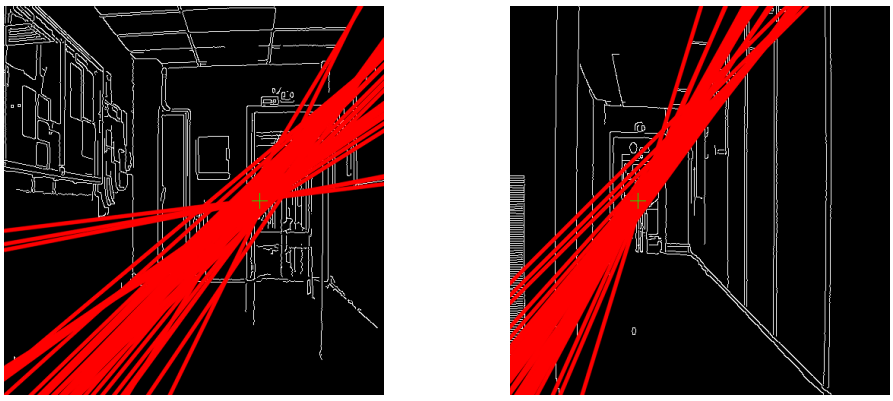
$$\rho_\theta = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta \tag{5}$$

Cada píxel de la imagen votará a aquellas líneas a las que pueda pertenecer. Obviamente, no todos los píxeles tendrán derecho a voto, si no sólo aquellos cuyo módulo del gradiente sea mayor de un umbral establecido. Una forma gráfica de verlo sería la expuesta en [2]. Para cada punto (x_0, y_0), se marcan las combinaciones (ρ, θ) válidas, es decir, que dan lugar a una posible línea que pase por ese píxel. Al determinar las posibles líneas de todos los píxeles con derecho a voto, se encuentran las intersecciones entre ellas, que determinan, en función de cuántas intersecten en ese punto, cuántos píxeles han votado por ella, para así hacer posible su posterior selección. Acto seguido se retienen las líneas votadas por más de un número de píxeles determinado, mientras que el resto son descartadas.

Otro aspecto relevante es el alto coste computacional que tendría que cada píxel votase en todas las posibles líneas que lo atreviesen, dependiendo tanto de la resolución de ρ en píxeles, como la de θ en radianes, y de las orientaciones a las que se permita votar a cada píxel. La propia función permite restringir el voto de los píxeles a orientaciones similares al gradiente en esos puntos, que debe ajustarse en función de la imagen que se busque tratar. Asimismo, se puede establecer la resolución de los anteriores, habiéndose empleado 1 y $\frac{1}{\pi}$ radianes respectivamente.

Partiendo de las líneas obtenidas, se debe obtener el punto de fuga. En una imagen ćomo las tratadas podría haber hasta 3 puntos de fuga. El primero, resultado de la intersección de las líneas verticales de los marcos de las puertas. El segundo y el tercero se derivan de las líneas horizontales de la imagen y de las que se propagan hacia la profundidad del pasillo respectivamente. Debido a la forma del pasillo, el que cuenta con mayor interés es el último de ellos, ya que incluso con ligeros movimientos de la cámara los dos primeros se encontrarían fuera de la imagen, por lo que decidimos centrarnos en el último.

En la primera versión, se eliminaban las líneas verticales y horizontales mediante el θ de las mismas. Las restantes se intersectaban con la línea horizontal marcada por el centro de la imagen. Una discretización del espacio en grupos de píxeles permitía que los votos no se diseminasen por píxeles cercanos debido a pequeñas variaciones en el θ de las mismas. El grupo con mayor número de puntos nos da el punto de fuga. No obstante, esta solución estaba limitada a únicamente la línea anterior, lo que provocaba que al más mínimo movimiento de la cámara fuese difícil o incluso imposible obtener el punto de fuga en el lugar correcto, por lo que se pasó rápidamente a la implementación del apartado opcional.



(a) `pasillo2.pgm` (b) `pasillo3.pgm`

Figura 4: Ejemplos de obtención del punto de fuga previo a la mejora

4. Modificaciones a la obtención del punto de fuga

Expuesta la problemática anterior, se buscaba eliminar la limitación espacial a la hora de obtener el punto de fuga, permitiendo modificaciones en la posición de la cámara.

Tras el filtrado de las líneas verticales y horizontales, nombrado en el apartado anterior, se implementó un algoritmo para obtener el nuevo de fuga. Concretamente, se recurrió a RANSAC, que durante un número n de iteraciones realiza el siguiente proceso:

1. Calcula el punto de intersección entre dos rectas aleatorias (si existe, si no, termina la iteración).
2. Se comprueba para cada línea si intersecta con una circunferencia con un radio ajustable (actualmente 20 píxeles) y centro en el punto de intersección de las dos anteriores. Se contabiliza el número de rectas que lo hacen, los *inliers*.
3. Si el número es mayor que el máximo obtenido hasta ese momento, se conserva como el nuevo punto de fuga.

Con el algoritmo anterior ya se obtenían buenos resultados, pero en ocasiones aparecía un problema adicional cuando múltiples rectas en direcciones similares y con orígenes muy cercanos se cortaban lejos del punto de fuga. Para solucionarlo, se añadió la restricción de que el ángulo en coordenadas polares de las dos rectas elegidas para determinar el posible punto de fuga en una iteración deba diferenciarse en un umbral de grados (actualmente 12 píxeles).

El procedimiento anterior se probó con un vídeo de un pasillo lográndose muy buenos resultados, que pueden observarse en el vídeo adjunto a la entrega.[1]

[1] Es necesario ajustar los parámetros dependiendo de la imagen tratada. En el programa entregado han sido ajustados para este vídeo en concreto.

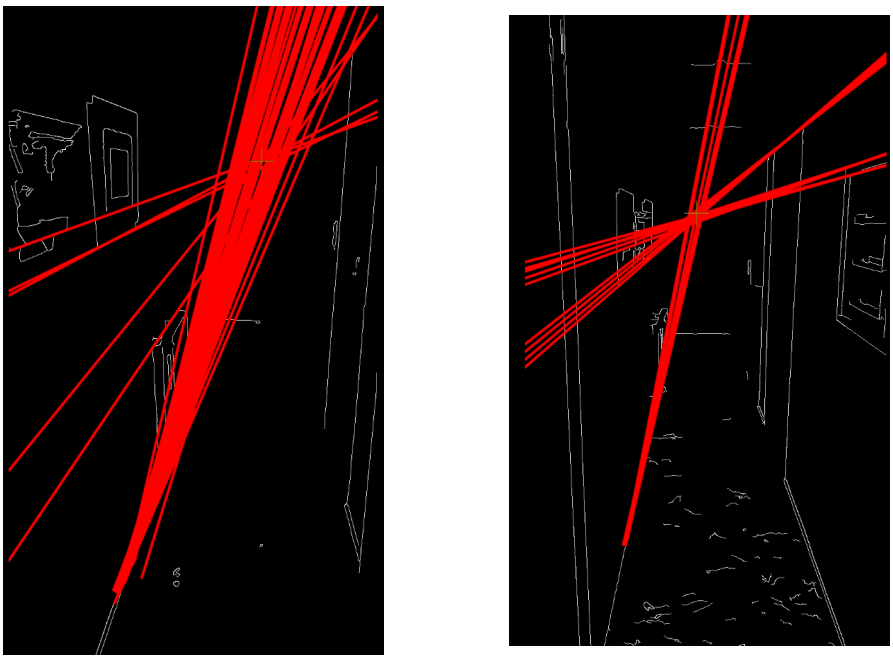


Figura 5: Ejemplos de obtención del punto de fuga

Referencias

[1] *Front-End Vision and Multi-Scale Image Analysis*, pages 53–69. 2003. http://www.sci.utah.edu/~gerig/CS7960-S2010/handouts/04%20Gaussian%20derivatives.pdf.

[2] Hough line transform. https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html. Consultado el 19/03/2021.