Práctica 2: Sistemes de Percepció

Percepció i Control per a Sistemes Encastats

Detección de contornos verticales de una puerta
Alumno: Dilpreet Singh

Para realizar esta práctica, he creado los siguientes dos scripts. A continuación, se describen en breve:

1- "Contorns.py": he creado este script para leer una imagen, convertirlo en escala de grises y su binarización posterior utilizando el umbral de Otsu. Por último, aplico el filtro de Canny para extraer los contornos de la imagen y lo exporta en el fichero de trabajo.

   Nota: si el usuario desea utilizar este script, tendrá que cambiar el código indicado el nombre de la imagen de entrada y el nombre de salida para cada imagen.

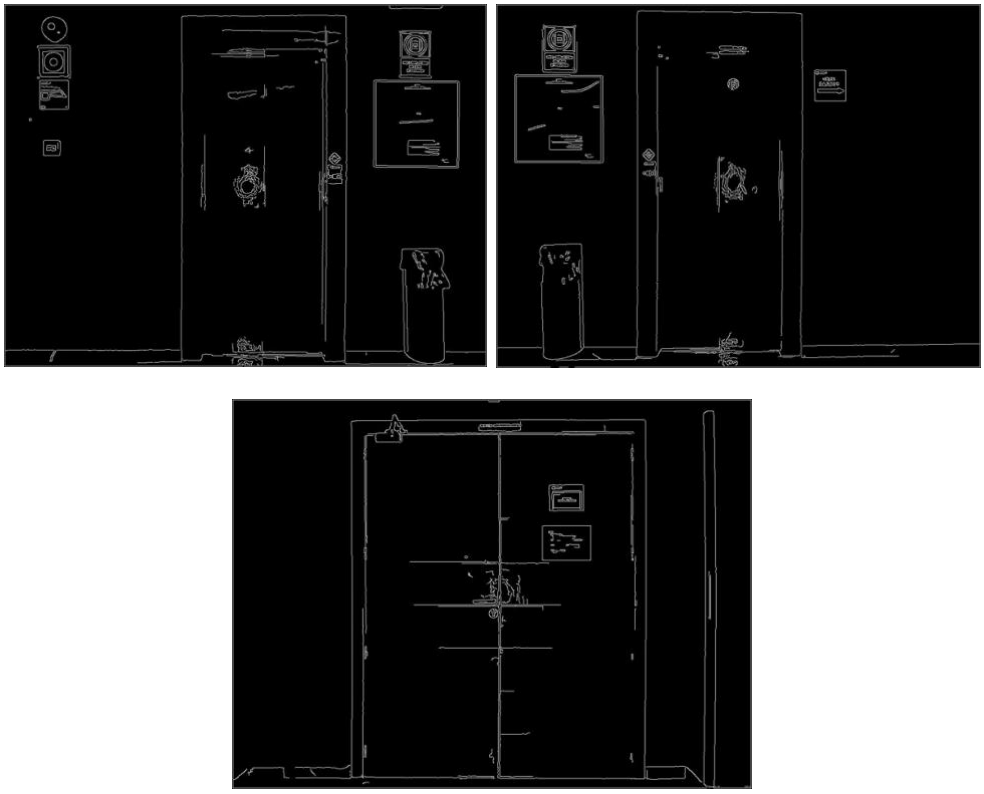
2- "Script.py": este script es el que a partir de la imagen de contornos detecta las líneas verticales de la puerta. Luego, dibuja las líneas sobre la imagen y los exporta en el mismo fichero de trabajo. En este script está implementada la Transformada de Hough.

Contorns.py

Antes de extraer los contornos de las imágenes sobre las cuales trabaja este script, he intentado mejorar la calidad de las fotos aplicando stretching del histograma o ecualización. Sin embargo, como no he visto grandes diferencias, he decidido no utilizar la técnica de stretching o ecualización.

Además, también he probado los filtros de Sobel, Roberts o Prewitt, pero como éstos generan contornos en color gris, en los pasos posteriores tenía problemas (transformada de Hough) para detectar píxeles blancos. Entonces, utilizo el filtro de Canny que presenta unos resultados bastante satisfactorios.

Imágenes resultantes de Contorns.py:



Script.py

Como he mencionado antes, “Script.py” es el más importante ya que tiene implementada la Transformada de Hough. Entonces, en la siguiente sección voy a describir este script.

Este script básicamente está estructurado mediante cuatro funciones:

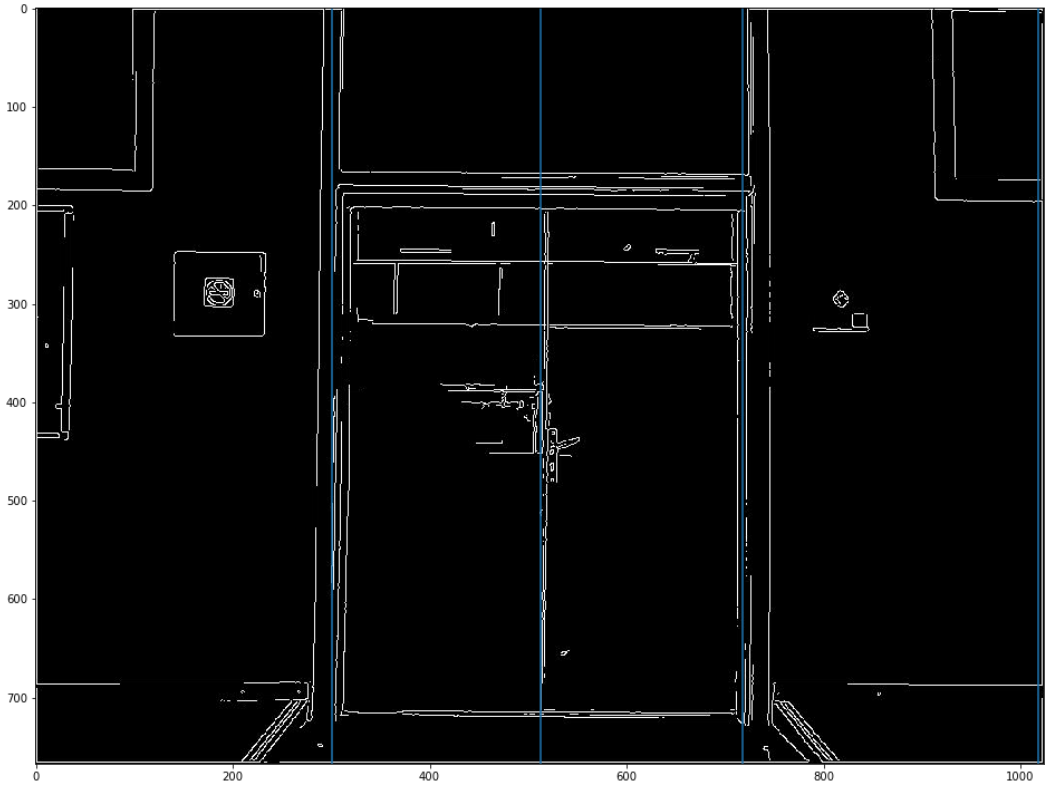
1- main() -> la función principal que inicia todo el proceso desde la lectura de las imágenes de contornos sobre las cuales se trabajará hasta la exportación de las imágenes finales.
2- get_x_intersection() -> devuelve un array que contiene los puntos de la eje X donde se tiene que dibujar las líneas verticales, es decir, los puntos de la eje X donde se cruzan la fila intermedia de la imagen y las líneas verticales de contornos de la imagen
3- hough() -> esta función implementa la transformada de Hough, recibe una imagen sobre la cual tiene que trabajar y al final devuelve la matriz de acumulación de Hough.
4- clear_cells() -> el código de esta función nos sirve para poner a cero N celdas vecinas de cada una de las líneas verticales que se dibujaran a partir de Ro y Theta de la matriz de acumulación de Hough.

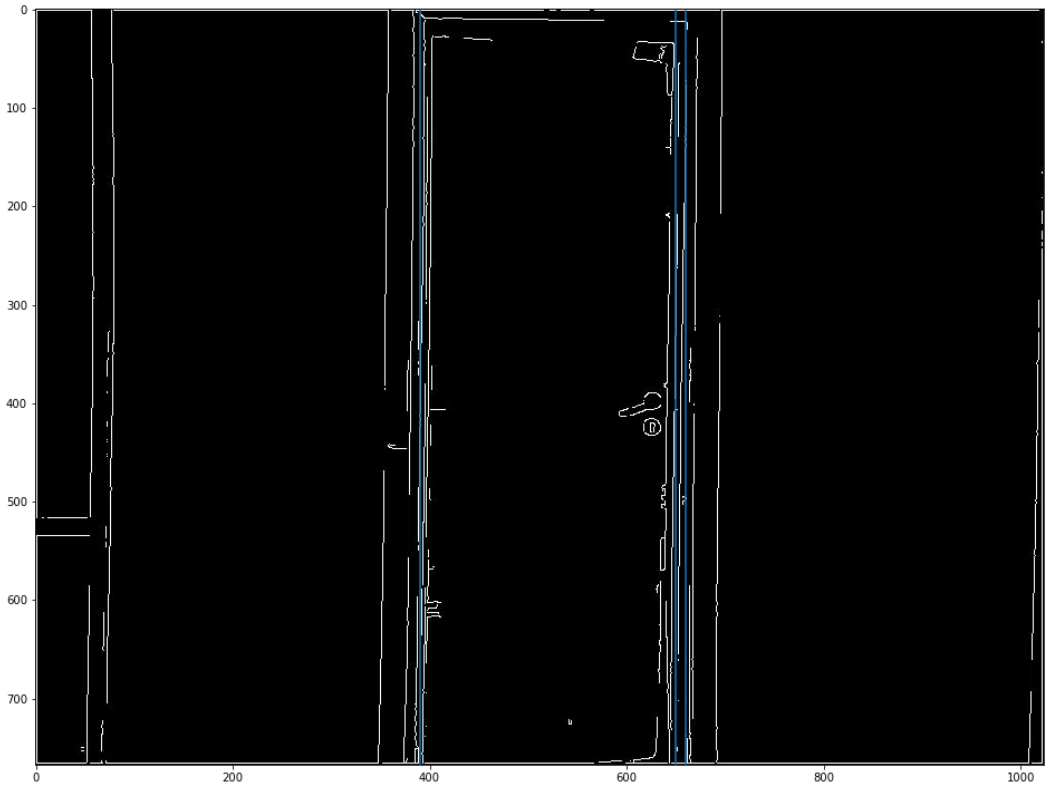
Para mantener la simplicidad de la documentación, describiré los pasos de forma detallada dentro del mismo código. Por ahora, explicaré los pasos principales que he tomado para implementar la transformada de Hough en la función hough():

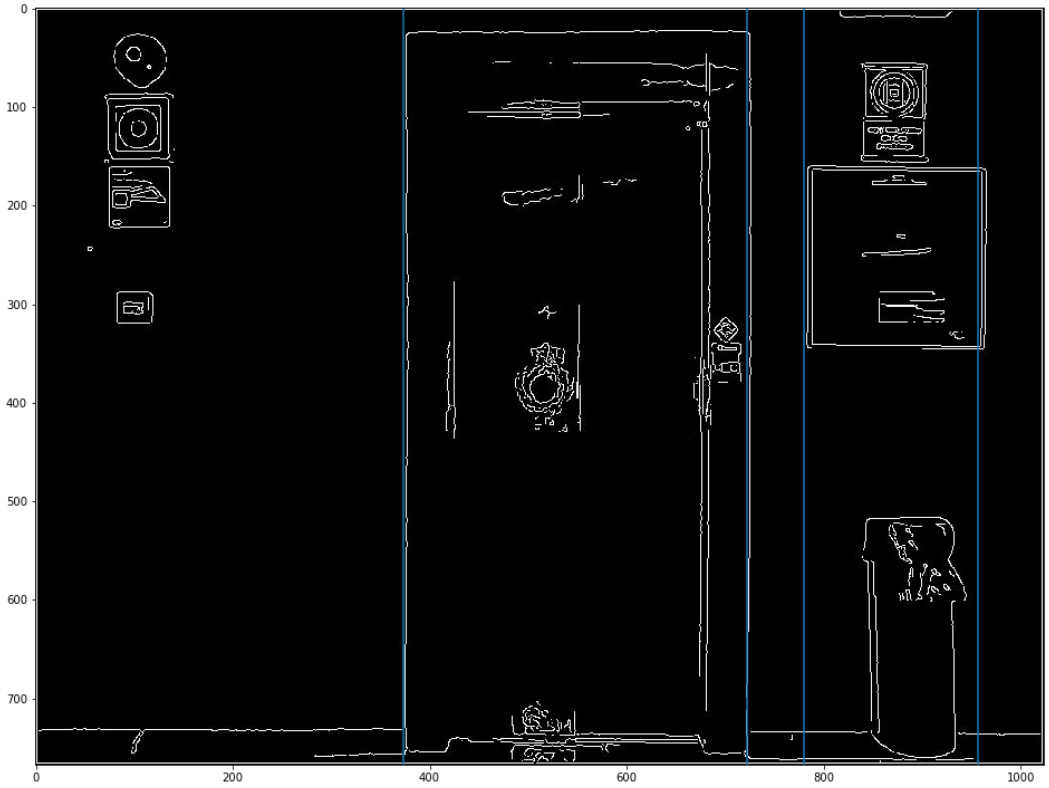
1. Obtener el valor máximo de Ro a partir de las dimensiones de la imagen, utilizando la fórmula de h^2 = x^2 + y^2. Ro máxima es la hipotenusa, x e y son la anchura y altura de la imagen respectivamente.
2. Theta mínima y máxima será de -10º y 10º.

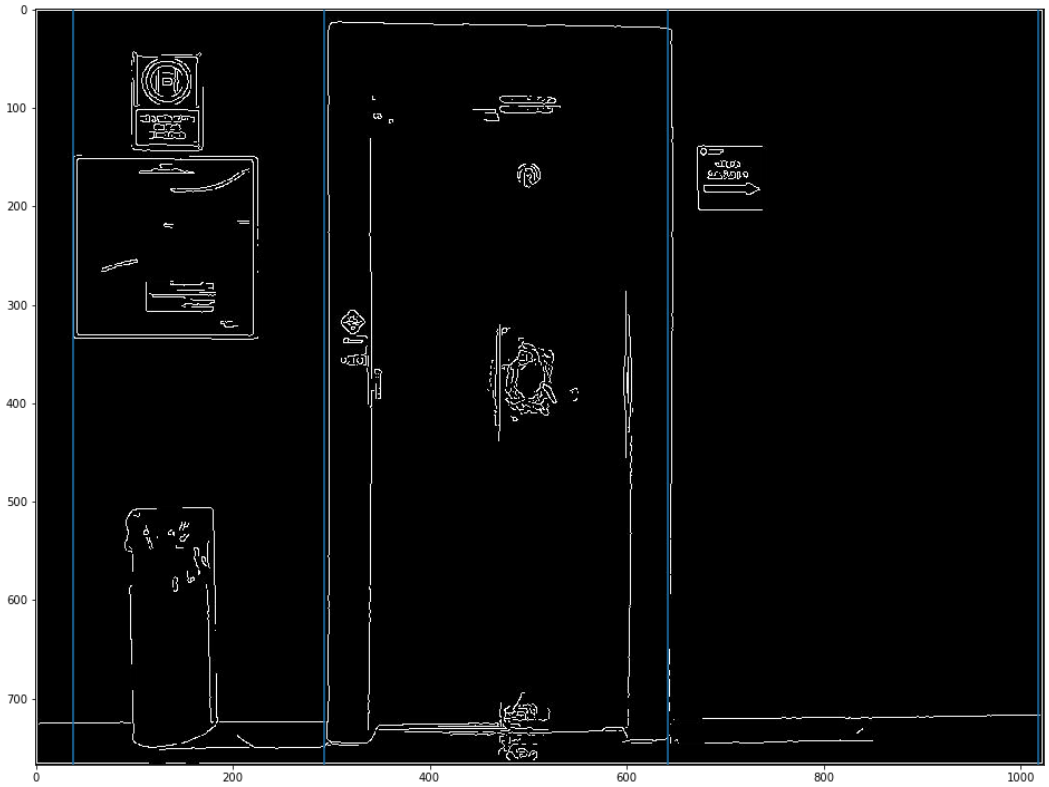
3. He puesto que el incremento del valor de Ro y de Theta será en una unidad (se puede cambiar dentro del código). Luego, he creado un array para los posibles valores de Theta y Ro (desde mínimo hasta máximo).
4. Inicializo una matriz con ceros de dimensión Ro_max*Theta_max.
5. Creo un listado de tuplas (x,y) de los píxeles en color blanco.
6. Para cada píxel blanco detectado, creo un bucle for para todos los valores de Theta y calculo el valor de Ro mediante la fórmula que está en las diapositivas de la clase. Además, utilizando los valores actuales de Ro y Theta, calculo los valores de S y T que me servirán para incrementar el valor en una unidad dentro de la matriz de acumulación.
7. Finalmente, cuando sale de los dos bucles, se devuelve la matriz.

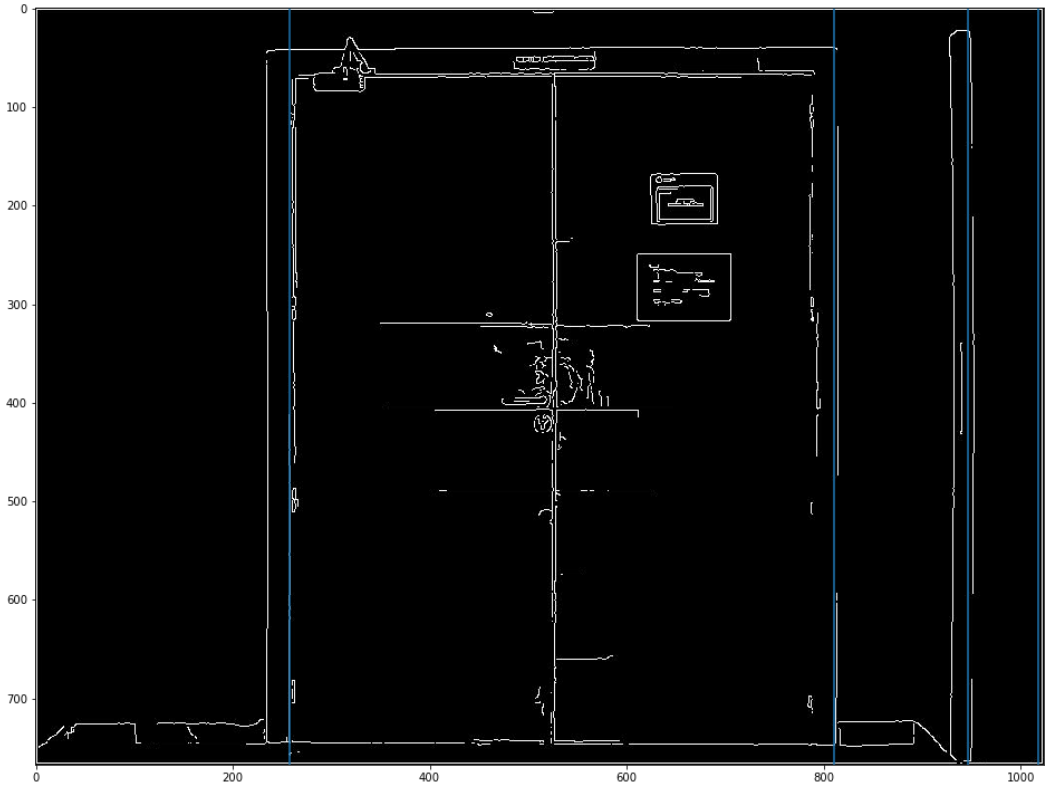
Imágenes resultantes del Script.py:




0
100
200
300
400
500
600
700
0
200
400
600
800
1000


0
100
200
300
400
500
600
700
0
200
400
600
800
1000


0
100
200
300
400
500
600
700
0
200
400
600
800
1000


0
100
200
300
400
500
600
700
0
200
400
600
800
1000