Práctica 4
Segmentación de la imagen

- Técnicas basadas en la frontera
- Umbralización
- Agrupación de píxeles

Robótica y Visión por Computador
Ing. Telecomunicaciones

Universidad Miguel Hernández

Objetivo de la práctica:

El objetivo de la práctica es estudiar ejemplos de las diferentes técnicas de segmentación de la imagen. Según lo estudiado en clase, podemos encontrar:

- Técnicas de segmentación basadas en la frontera
- Técnicas de segmentación por umbralización
- Técnicas de segmentación por agrupamiento de píxeles

En la práctica se va a utilizar la imagen *building.jpg* que se encuentra disponible en la web. Esta imagen se copia en el subdirectorio *\images* dentro de la carpeta *\practica4*. Se trata de una imagen a color de 868x600 pixeles.

Realización de la práctica:

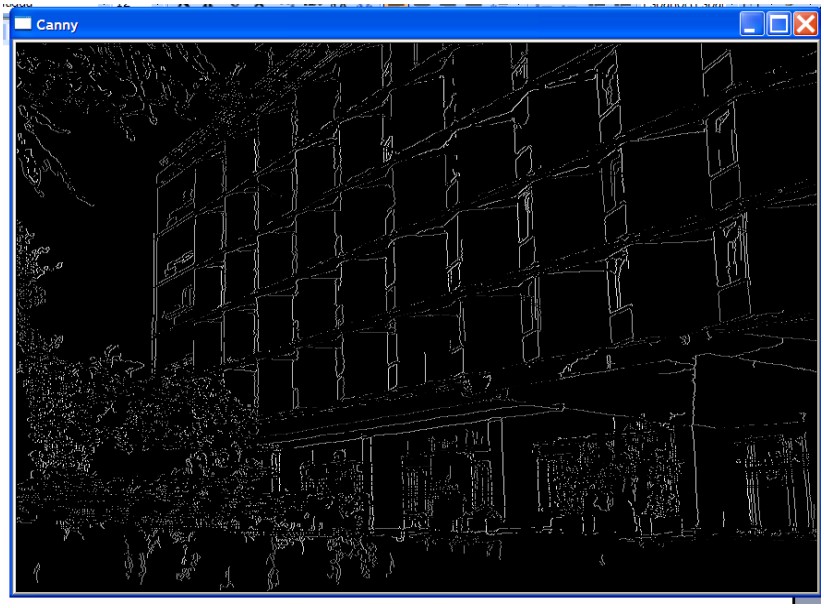
1. En primer lugar se debe abrir el workspace creado en la práctica 0 o bien generar uno nuevo tal y como se explicó en dicha práctica.
2. Añadimos un nuevo proyecto al workspace, que llamaremos **practica4**
    a. Si estamos trabajando sobre el workspace anterior, los proyectos `cv.dsp, cvaux.dsp` y `highgui.dsp` ya están añadidos, y lo único que debemos hacer es establecer las dependencias y los settings del nuevo proyecto creado.
    b. Si hemos creado un proyecto nuevo, se deben añadir los proyectos `cv.dsp, cvaux.dsp` y `highgui.dsp` tal y como se explicó en la práctica 0 y luego establecer la configuración adecuada.
    c. Desde el explorador de Windows, creamos un directorio `\images` dentro de la carpeta `\practica4` y colocamos la imágen *building.jpg*.
3. Inicializar las siguientes estructuras que serán utilizadas en la práctica:

```
IplImage* src = NULL;// imagen fuente
IplImage* dst= NULL; //imagen destino intermedia. Utilizada para canny
IplImage* contornos=NULL; //imagen destino utilizada para los contornos
IplImage* color_dst= NULL; //imagen destino utilizada para Hough
IplImage* umbral= NULL; //imagen destino utilizada para threshold
CvMemStorage* storage_con = cvCreateMemStorage(0); //para almacenar los contornos
 CvSeq* lines_con = 0;
CvMemStorage* storage = cvCreateMemStorage(0); //para almacenar las lineas de
Hough
 CvSeq* lines = 0;
```

4. Abrir la imagen fuente como una imagen de grises (Ver el parámetro `iscolor` de la función `cvLoadImage`)
5. Crear la imágenes `dst, contornos, umbral` como imágenes monocanal y la imagen `color_dst` como imagen de 3 canales.
6. Escribir el siguiente código

```
cvCanny( src, dst, 50, 200, 3 );
cvCvtColor( dst, color_dst, CV_GRAY2BGR );
```

Con este código almacenamos en la imagen `dst` los contornos obtenidos por el algoritmo de Canny, y también lo hacemos en la imagen `color_dst`.



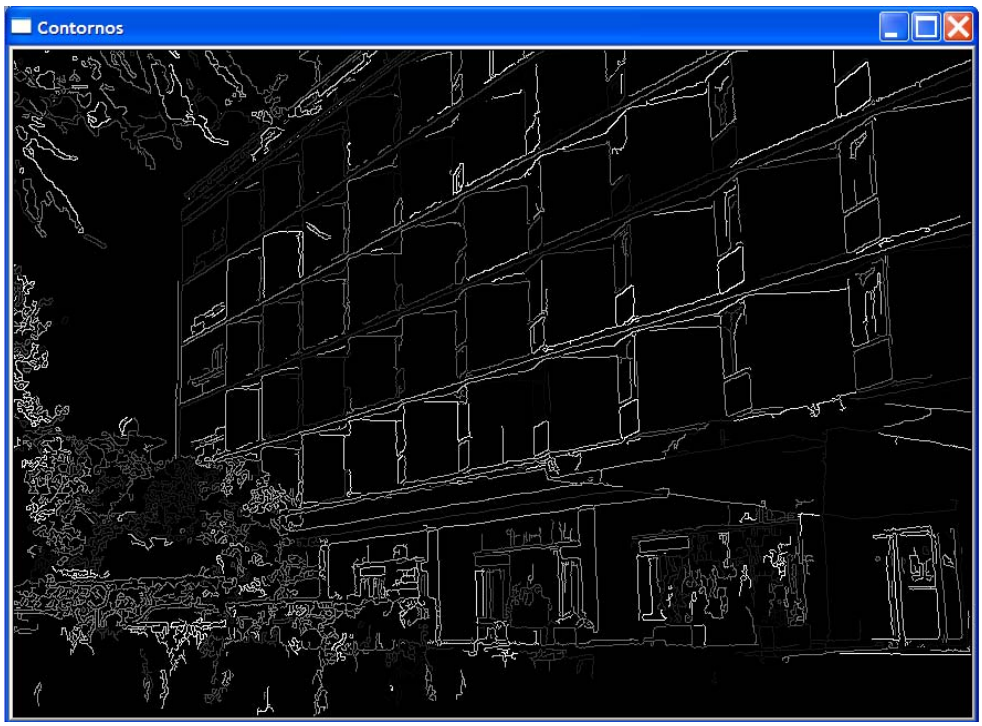
CONTORNOS----------

7. Para la detección de contornos vamos a probar con la función cvFindContours. Escribimos el siguiente código:

```
        cvFindContours(dst,storage_con,&lines_con, sizeof(CvContour),CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE,cvPoint(0,0));
        cvZero(contornos);
        for( ; lines_con != 0; lines_con = lines_con->h_next )
         {
         CvScalar color = CV_RGB( rand()&255, rand()&255, rand()&255 );
        cvDrawContours(contornos,lines_con,color,color,-1,CV_FILLED,8,cvPoint(0,0));
         }
```

NOTA: la imagen `dst` *utilizada es la imagen binaria obtenida por Canny. También se puede obtener con un Threshold binario, como veremos posteriormente.*

Mostrando la imagen obtenida (`contornos`) debemos observar una figura como la siguiente:



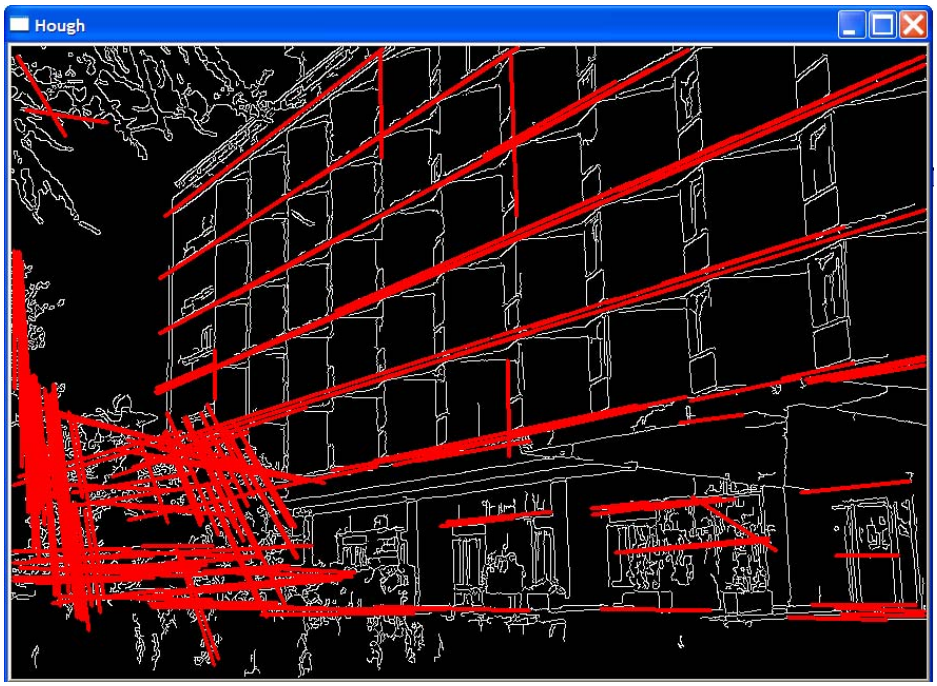
8. Para estudiar la transformada de Hough para líneas rectas vamos a trabajar con la función `cvHoughLines2`. A continuación se muestra como ejemplo el código para usar esta función con las imágenes de la práctica. Observar el comportamiento de los parámetros de la función tanto para el caso de `CV_HOUGH_STANDARD` como de `CV_HOUGH_PROBABILISTIC`.

```
/*//PARA HOUGH STANDARD
        lines = cvHoughLines2( dst, storage, CV_HOUGH_STANDARD, 1, CV_PI/180, 100, 0, 0
);
    for( i = 0; i < MIN(lines->total,100); i++ )
    {
        float* line = (float*)cvGetSeqElem(lines,i);
        float rho = line[0];
        float theta = line[1];
        CvPoint pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a*rho, y0 = b*rho;
        pt1.x = cvRound(x0 + 1000*(-b));
        pt1.y = cvRound(y0 + 1000*(a));
        pt2.x = cvRound(x0 - 1000*(-b));
        pt2.y = cvRound(y0 - 1000*(a));
        cvLine( color_dst, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
}*/

//PARA HOUGH PROBABILISTIC
        lines = cvHoughLines2( dst, storage, CV_HOUGH_PROBABILISTIC, 1, CV_PI/180, 80,
30, 10 );
    for( i = 0; i < lines->total; i++ )
    {
```

```
    CvPoint* line = (CvPoint*)cvGetSeqElem(lines,i);
    cvLine( color_dst, line[0], line[1], CV_RGB(255,0,0), 3, 8, 0 );
}
```

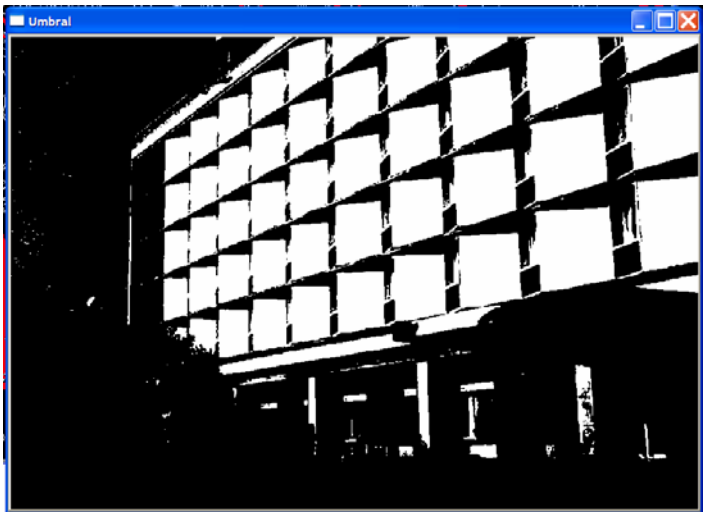
Con los parámetros del ejemplo debemos obtener la siguiente figura para el caso de `CV_HOUGH_PROBABILISTIC`.



UMBRALIZACION----------

9. Vamos a utilizar la función `cvThreshold` para comprobar los procesos de umbralización. Se pide comprobar los diferentes parámetros de esta función. Probar con las diferentes configuraciones:

```
cvThreshold(src,umbral,200,255,CV_THRESH_BINARY);
cvThreshold(src,umbral,200,255,CV_THRESH_BINARY_INV);
cvThreshold(src,umbral,110,255,CV_THRESH_TRUNC);
```



AGRUPACIÓN DE PIXELES (TODO)----------

10. Liberar memoria de todas las imágenes utilizadas.
11. Anotar para cada caso anterior la conclusión obtenida.