22437 - Industrial Vision
Lab 7: Edge and Line Detection

Jaume Taberner Ferrer

Universitat de les Illes Balears

Useful functions: *imfilter*, *fspecial*, *mat2gray*, *edge*, *im2bw*, *hough*, *houghpeaks*, *houghlines*

An edge is a set of connected pixels which lie on the boundaries between two regions, which typically correspond to two different objects or parts of objects. Assuming planar objects, edges correspond to abrupt intensity changes. Such discontinuities can be detected by using first- and second-order derivatives.

First-Order Derivatives

For two-dimensional functions, the concept of derivative is associated with the *gradient*. It is defined as the vector:

$$\nabla f(x,y) = (G_x, G_y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y} \right)$$

The magnitude (module) of this vector is:

$$|\nabla f(x,y)| = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$$

The gradient vector points in the direction of the maximum rate of change of f at coordinates (x, y). This direction is computed as:

$$\angle \nabla f(x,y) = \arctan\left(\frac{G_y}{G_x}\right)$$

The gradient is approximated digitally by means of spatial filtering, with appropriate convolution masks:

Roberts: (cross-diagonal)

$\nabla_v = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ $\nabla_u = \begin{bmatrix} -1 & 1 \end{bmatrix}$ $\nabla_v = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ $\nabla_u = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

Prewitt: $\nabla_v = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ $\nabla_u = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

Sobel: $\nabla_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ $\nabla_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

For detecting edges using first-derivatives, we have to compute the gradient of the image in both directions and establish a threshold according to the magnitude at each pixel. Pixels with magnitudes higher than this threshold are considered as an edge.

Perform the following tasks:

1. Write a function in Matlab to compute the gradient of an image in both directions using the Sobel operator. You can use Matlab functions, except *edge*. The signature of the function must be:

```
function [Gx, Gy] = gradients(img)
```

where *Gx* and *Gy* are the gradients and *img* is an image with values between the range [0.0, 1.0].

2. Load and display the image *house.jpg*.

3. Compute the gradients of the image in both directions using the function created in exercise 1. Display the resulting gradients, converting them to a gray scale image.

4. Compute the magnitude of the gradient at each point using the above-mentioned approximation and display the final magnitude image.

5. Detect edges in the image, selecting as edges pixels whose magnitude is above a given threshold. Try different values for this threshold and display the results.

6. Using the *edge* Matlab's function, compute the edges of the image using Sobel, Prewitt and Roberts operators. Test several thresholds for each operator and display the best results obtained. Compare them, enumerating the main advantages and disadvantages of each approach.

7. Apply the Canny edge detector to the image using the default parameters, display the obtained edges and compare the results with the previous ones. Vary the parameters of the algorithm and explain the observed effects.

Second-Order Derivatives

Second-order derivatives in image processing generally are computed using the Laplacian operator. A way of detecting edges using second-order derivatives is to find places where the Laplacian response is passing by zero. The Laplacian seldom is not used directly for edge detection because it is sensitive to noise. It is usually combined with a Gaussian function in order to perform a smoothing previous operation over the image. This is known as the Laplacian of Gaussian (LoG), or Marr-Hildreth operator.

Perform the following tasks:

1. Write a function in Matlab to compute zero-crossings given a second-order derivative matrix. The signature of the function must be:

```
function imedges = zerocrossings(deriv2, T)
```

   where *deriv2* is a second-order derivative image, *T* is a threshold for the magnitude and *imedges* is a binary image with values established to 1 where exists a zero-crossing. A position (x, y) will be considered as zero crossing if its sign changes in relation to the pixels immediately to the right or below with a magnitude above some threshold T.

2. Reload and display the image *house.jpg*.

3. Compute the second-order derivative of the image using the Laplacian operator. Display the results.

4. Compute the edges of the image using the derivative obtained in the previous exercise and the function coded in exercise 1. Apply several thresholds and display the output images.

5. Compute the second-order derivative of the image using a LoG filter of size 13x13 and sigma 2.0. Display the results.

6. Compute the edges using the results of the previous exercise and the function written in exercise 1. Apply several thresholds and display the resulting images.

7. Compute the edges of the image using the two previous operators by means of the *edge* Matlab's function, with default parameters. Display the results and compare them with the previous ones.

Hough Transform

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform. The classical Hough transform was concerned with the identification of lines in the image, but later the Hough transform has been extended to identifying positions of arbitrary shapes, most commonly circles or ellipses.

Using the functions available in the Matlab's Image Processing Toolbox, employ the Hough transform to detect the lines of the road present in the image *road.jpg*. As a final result, you should plot the accumulator and the detected lines in the original image as shown in the following figure: