Universitat Politècnica de Catalunya
Universitat de Barcelona
Universitat Rovira i Virgili

Master in Artificial Intelligence

Computer vision

Edges and contours

Authors:
Johannes Heidecke
Alejandro Suárez Hernández

October 2016

Contents

List of Figures

1 Determine the optimal edges

We have considered seven different detectors:

- Simple derivative and thresholding method
- Sobel derivative and thresholding with NMS (Non-Maximum Suppression). This method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of I is maximum. Returns all edges that are stronger than a certain threshold. The NMS takes care of thinning the edges.
- Prewitt derivative and thresholding with NMS. Same than Sobel, but with the Prewitt approximation of the derivative instead.
- Roberts derivative and thresholding with NMS. Same than Roberts, but with the Roberts approximation of the derivative instead.
- LoG (Laplacian of Gaussians). This method performs the convolution of the image with an approximation of the Laplacian of a Gaussian function. There is also a threshold involved with a similar role than in the previous methods.
- Zero-cross method. This method finds edges by looking for zero-crossings after filtering I with an arbitrary filter. A threshold may be specified.
- Canny. The Canny method finds edges by looking for local maxima of the gradient of I. The edge function calculates the gradient using the derivative of a Gaussian filter. This method uses two thresholds to detect strong and weak edges.

The simple derivative detector has been implemented by us as follows:

$$D_x = [-1\ 1] \quad D_y = [1\ -1]^T \tag{1a}$$

$$B_x = I * D_x \quad B_y = I * D_y \quad B^{\circ 2} = B_x^{\circ 2} + B_y^{\circ 2} \tag{1b}$$

where $|\cdot|^{\circ 2}$ denotes the element-wise square of a matrix.

$$I_{edges} = B^{\circ 2}_{\geq T^2} \tag{1c}$$

where the $A_{\geq t}$ denotes a matrix with 1 in each position (i, j) where $a_{ij} \geq t$, and zero otherwise. Notice that T is the threshold and that we do not perform NMS.

We have used the MATLAB implementation of the other methods (the implementation that comes with the *edge* function).

As for which is the best edge detector, there is not close answer. It highly depends on the application and the evaluation criteria.

XX Sobel: This method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of I is maximum. Returns all edges that are stronger than threshold

XX t can be estimated automatically, for example simply using the default Sobel mode results in a t of 0.3715 for this image. Manual experiments with changing the value of t result in much better results for t = 0.1. There are many possible t values in the range 0.05 to 0.2 that yield almost same results. imabsdiff only shows a few pixel difference.

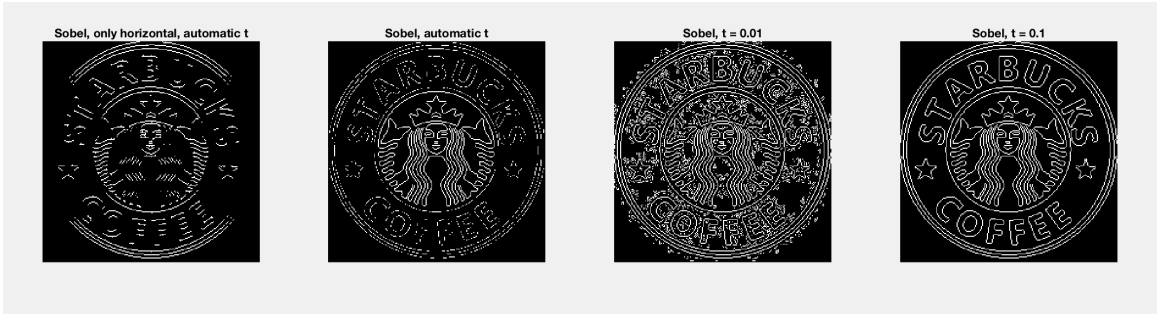

Figure 1: Results with Sobel method

XX Prewitt: This method finds edges using the Prewitt approximation to the derivative. It returns edges at those points where the gradient of I is maximum. Returns all edges that are stronger than threshold

XX Prewitt: automatic t is estimated at 0.3629. Much better results with a lower t of 0.25. A higher t e.g. 0.4 yields in the outer edges to not be detected anymore.

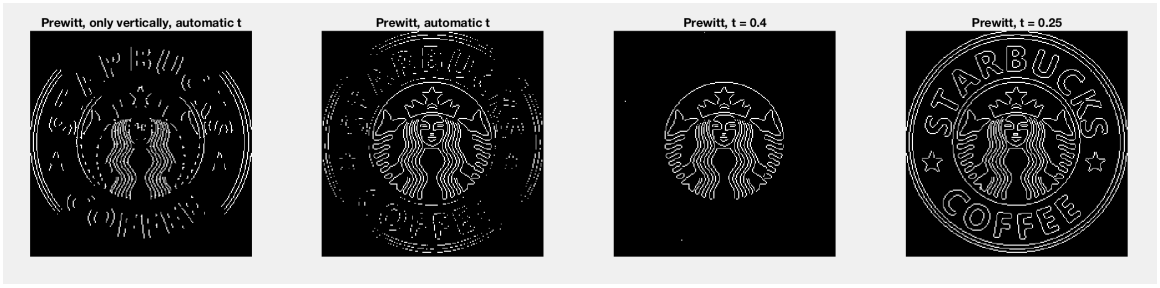

Figure 2: Results with Prewitt method

XX Roberts: This method finds edges using the Roberts approximation to the derivative. It returns edges at those points where the gradient of I is maximum.

XX Roberts automatic t = 0.4100

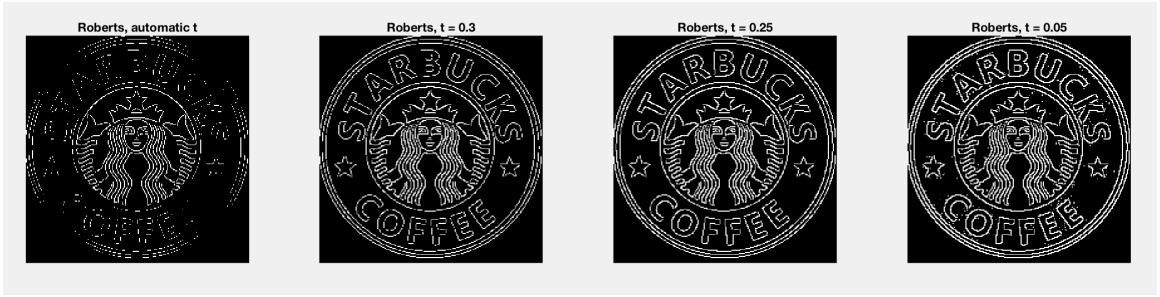

Figure 3: Results with Roberts method

XX log: This method finds edges by looking for zero-crossings after filtering I with a Laplacian of Gaussian filter. return all edges that are stronger than threshold.

XX automatic t: 0.022. Standard sigma is 2. Second image: change sigma to 1. Size of the gaussian filter is determined automatically as *n=ceil(sigma*3)*2+1* Higher t worsens result. t can be much smaller and still yield the same results.

XX zerocross: This method finds edges by looking for zero-crossings after filtering I with a filter that you specify, h. The edge function returns edges that are stronger than threshold.

Figure 4: Results with Laplacian of Gaussian

XX for this example, after some experiments, a gaussian filter with size 5x5 and sigma 2 was chosen. The automatic t for this filter was estimated at 0.0093. Increasing this t leads to important edges not being detected anymore. Lowering the threshold produces more false positives while some edges are still not being detected. The filter adds more dimensions of complexity for determining optimal parameters. While it might be possible to accomplish very good results with the right parameters, this method seems to require a lot of manual work, since the automatic estimation of the parameters does not yield good results.

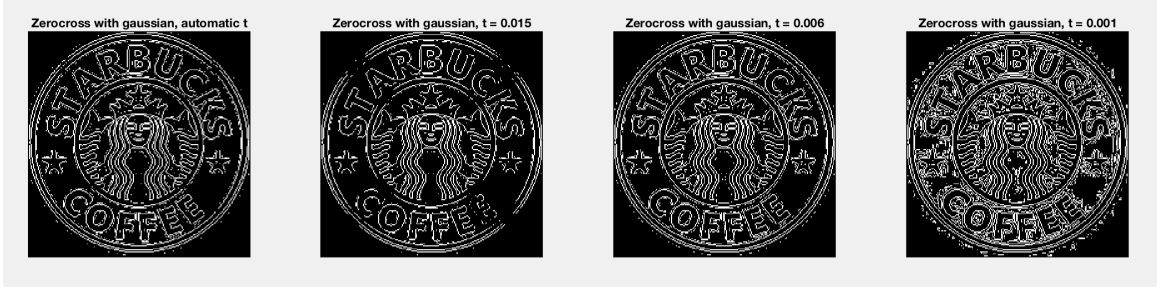

Figure 5: Results with Zerocross method

XX Canny: the Canny method finds edges by looking for local maxima of the gradient of I. The edge function calculates the gradient using the derivative of a Gaussian filter. This method uses two thresholds to detect strong and weak edges, including weak edges in the output if they are connected to strong edges. By using two thresholds, the Canny method is less likely than the other methods to be fooled by noise, and more likely to detect true weak edges. Disadvantage: not supported to be executed on GPU.

XX t: 0.1875 0.4688. Default: weak edge threshold is 0.4 of strong t.

Figure 6: Results with Canny method

XX: answer questions 2 & 3

XX: describe overlapping edges and images

XX: describe effects when code is run with other images, difficulties, etc.
XX answer questions 4, 5

2 Enhancing images with edges

Appendices

A Delivered code files

Here we list all the delivered script and function files delivered for this practice. We include relevant observations. For full insight, we refer the reader to the source code.

- `exercise1.m`: version of *exercise1* with UI controls
- `exercise1.m.old`: first version of *exercise1*, with hardcoded parameters
- `exercise2.m`: method that plays a video with overlapped edges. The edge detector can be configured at runtime.
- `exercise2.m.old`: method that extracts a single frame from a video and extracts its edges using several detectors. The edges are then compared.
- `overlapEdges.m`: auxiliary function that overlaps edges to an image. The color of the edges can be specified.
- `rndColor.m`: auxiliary function for generating a random color. Used to decide a different color for the overlapped edges each time.