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

1.1 Considered detectors

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 to test the other methods (available through the *edge* function).

In order to perform our experiments, we have implemented an interface with interactive controls so the user can select the detector and its parameters in runtime `exercise1.m`. An example of these controls can be seen in figure 5. There is also a more classical script with the name `exercise1.m.old`.

1.2 Comparison of methods

As for which is the best edge detector, there is not close answer. It highly depends on the application and the evaluation criteria. For our analysis, we consider mainly these four aspects: general performance, how good the edge detection is when we let the method automatically computes its parameters; the number of parameters in case there is need to fine-tune the method; and the efficiency. The last of these qualities could be measured thanks to the experiments in exercise 2.

The Canny detector is the most sophisticated detector and has a very good performance for a wide domain of images. It yields very good results when we let the detector choose automatically the threshold parameters and with the default $\sigma = 1$. The most obvious drawbacks are that it is the most time-consuming method and that, when the default parameters do not work good enough, we have to fine-tune three parameters (the high threshold, the low threshold and the σ value) for our particular application. Even so, a fixed set of parameters can give outstanding results for some applications. Results can be seen in figure 1.

Figure 1: Results with Canny method

The Sobel, Prewitt and Roberts detector lead to very similar result. The main difference among them is how they approximate the derivative. The Roberts detector has the particularity of not being useful for calculating the horizontal/vertical edges, because it uses approximation of the diagonal derivatives instead of the derivatives along the main axis. These detectors are quite fast, and in some cases they give reasonable performance when we let the detector find the threshold automatically. Usually it is necessary to tweak the threshold to obtain results more akin to our perception of the edges. An advantage with respect to the Canny detector is that the method takes only one parameter (if we let the direction of the edges aside). We can see some results in figures 2, 3 and 4.

Figure 2: Results with Sobel method

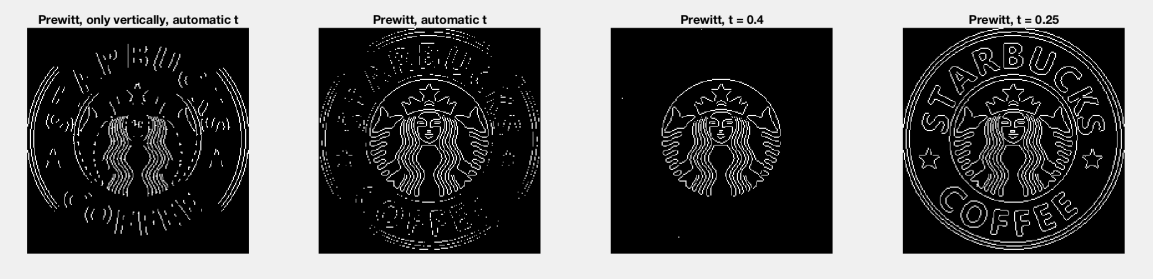

Figure 3: Results with Prewitt method

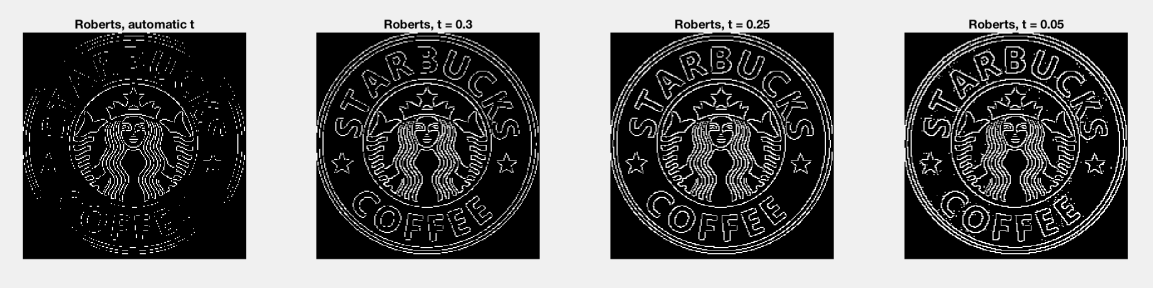

Figure 4: Results with Roberts method

The simple derivative detector did not provide any clear advantage over the previous filters neither in time nor in efficiency. Moreover, we do not perform NMS to thin the edges and it is pretty vulnerable to noise. The only advantage is that it is very simple to implement, even in low-level languages like C or ASM. In practical applications, this is the only reason why this method might be preferred over any of the previous ones. Much like Sobel, this method takes just one parameter. It is worth mentioning that we have also implemented a strategy for automatically selecting the threshold. This automatic threshold selection consists in taking 2 times the average of $B^{\circ 2}$ (i.e. in terms of signal processing, 2 times the *power* of B). Graphical results can be seen in figures 5 and 6.

The LoG method provides good results when using the automatic threshold and the default σ. However, it has the drawback of being too reactive to changes in the input (i.e. for a fixed σ, a small increment of the threshold can be the difference between clearly extracted edges and a poor edge selection). In terms of efficiency, it is between the Sobel

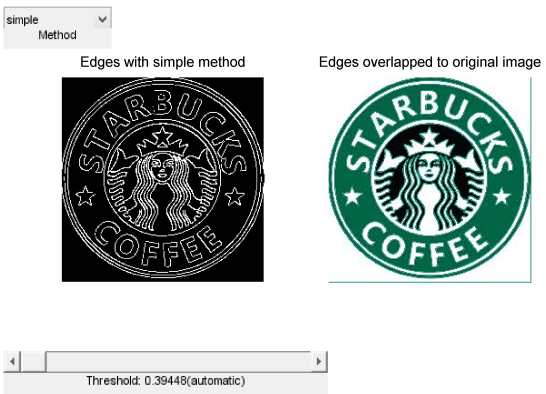

Figure 5: Simple detector with arbitrary threshold. Observe the thick edges.

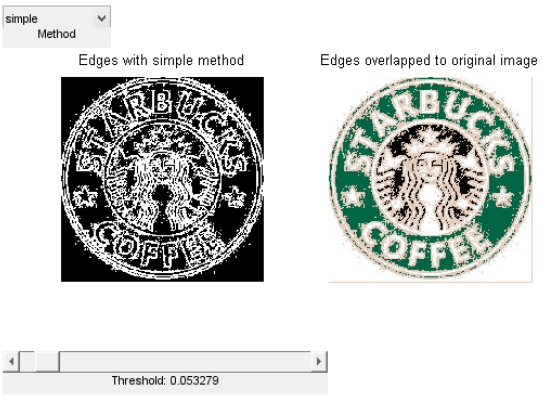

Figure 6: Simple detector with arbitrary threshold. Observe the noisy behavior.

and the Canny detector. In case that the automatic selection of the threshold and the default σ are not good enough, there are two parameters two tune, which is also between the Sobel and the Canny method. See some graphical examples of this edge detector in figure 7.

Figure 7: Results with Laplacian of Gaussian

Lastly we have the zero-cross method. In our experiments it had a performance very similar to that of the LoG method (with less abrupt differences when slightly changing the parameters). Also like LoG, its execution time was between Sobel and Canny. The main particularity of this method is that we get to choose the filter it applies to the image before computing the edges, so the number of parameters actually depends upon the filter we choose. We have chosen a Gaussian kernel with variable σ and a kernel size of at least 5 times this σ, so all in all the method has two parameters (the threshold and the σ). We can see some results in figure 8.

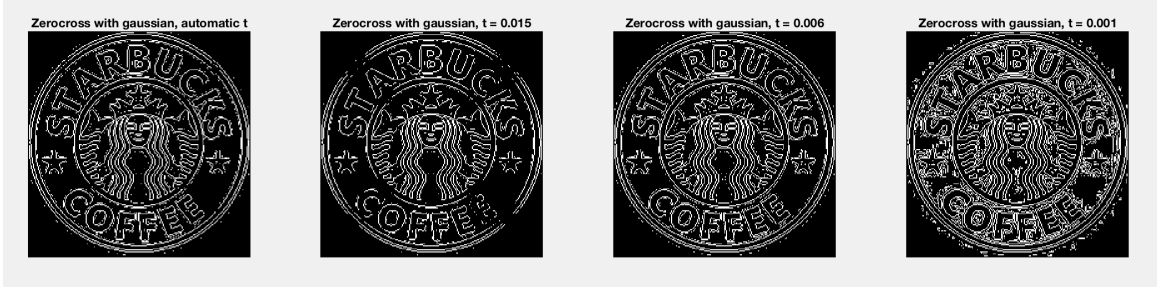

Figure 8: Results with Zerocross method

1.3 Selecting a detector and tuning its parameters

For this section we assume, based on the previous points, that the Canny detector has the potential to be the best detector provided that the application allows to tweak its parameters. The edges of the *Starbucks* logo can be detected almost perfectly with the automatic threshold and $\sigma = 1$, as it is shown in figure 9.

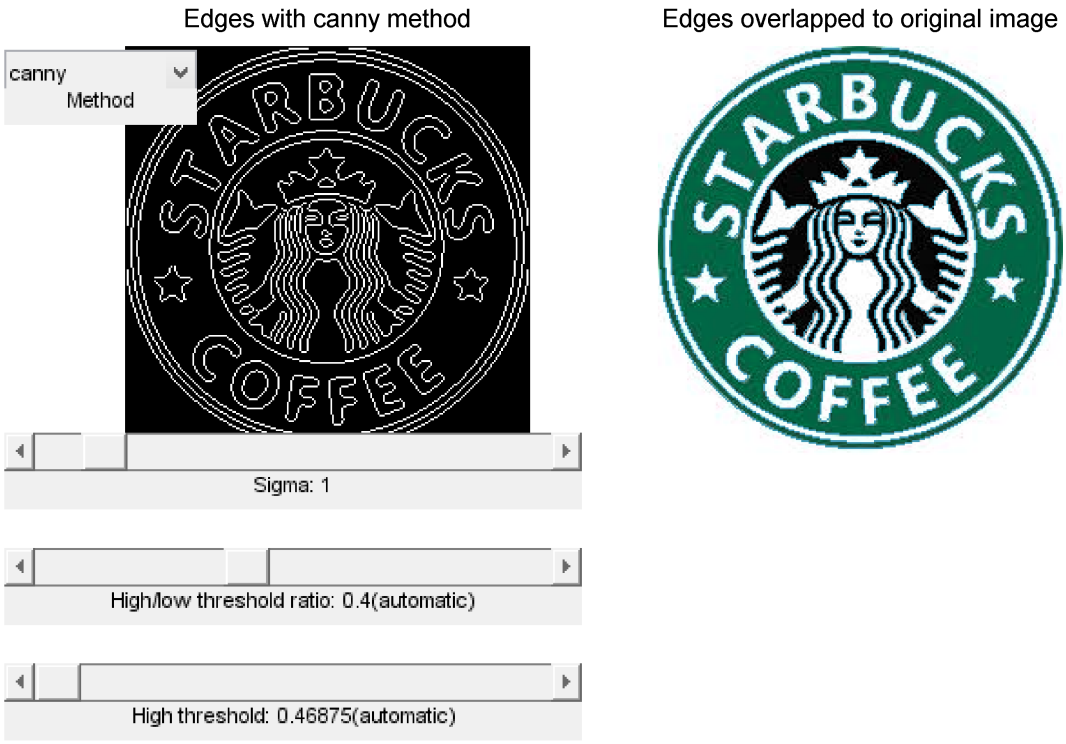

Figure 9: Selected parameters for extracting the edges from the Starbucks logo with Canny

However, if we try to extract the edges of another image with the same parameters (i.e. using the same thresholds), we obtain what we can see in figure 10.

For this same image, if we let the detector choose the threshold automatically, we obtain the edges in figure 11. In this case, the automatic threshold gives a reasonable result.

Even so, we might prefer fewer and simpler edges. For this, we need to manually tune the parameters of the Canny detector. Modifying both the high threshold and the σ value, we have obtained the edges of figure 12, which is arguably the best of the three results we have shown so far for this image.

This leads to the conclusion that there does not exist an universal set of parameters that yield good results for every image. In many cases, to achieve optimal results we would need to manually tweak the parameters.

We could also need a highly adaptable edge detector for many images (for example, to extract the edges from a video). In such circumstance, it is reasonable to let the detector choose automatically the threshold parameters and to choose the σ according to the resolution. Alternatively, we could set a fixed set of parameters if we believe that they will be good enough for our particular application.

Although for simplicity we have detailed here just the Canny detector, our experiments have lead to the same conclusion for the other methods as well.

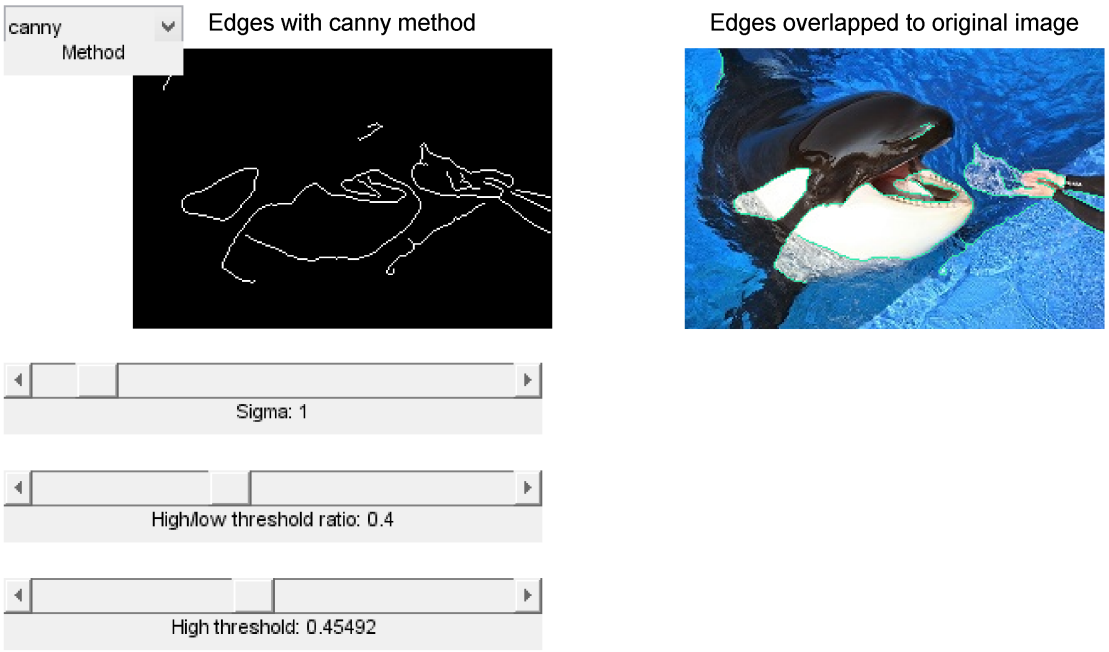

Figure 10: Extraction of the edges from `doulphin.jpg`. Observe that the threshold is approximately the same than the used for the edge extraction in the Starbucks logo. There are only a few visible edges.

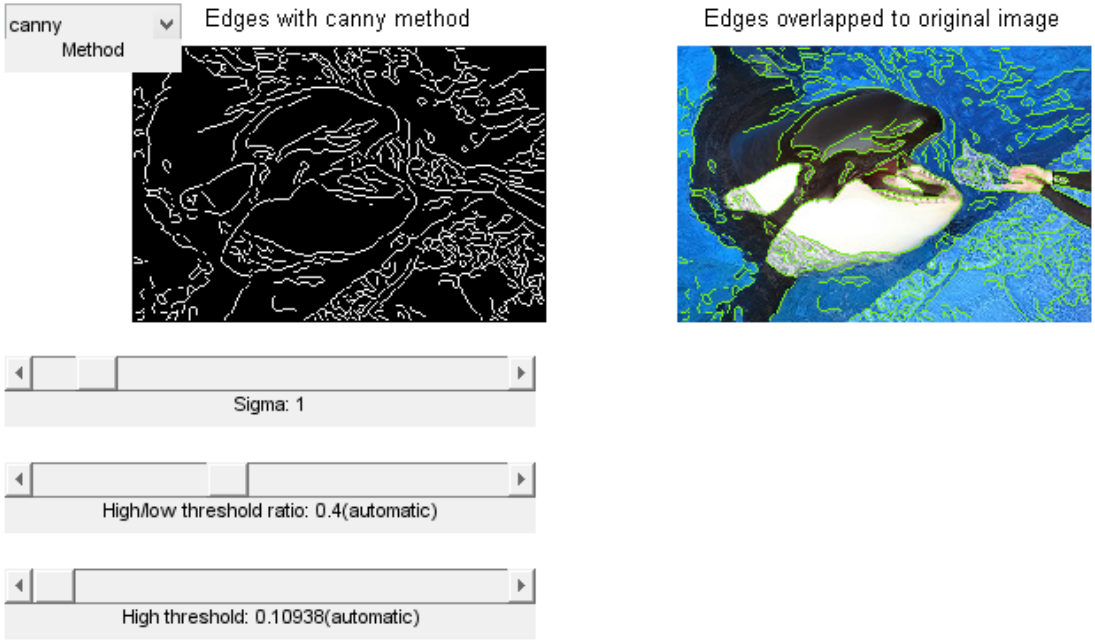

Figure 11: Extraction of the edges from `doulphin.jpg` using automatic thresholds. The quality is reasonable. However, we might prefer a less cluttered image.

2 Enhancing images with edges

Our function is capable of playing the video and overlap the detected edges. For this exercise we have embedded some UI controls, so the user can change the edge detection method and

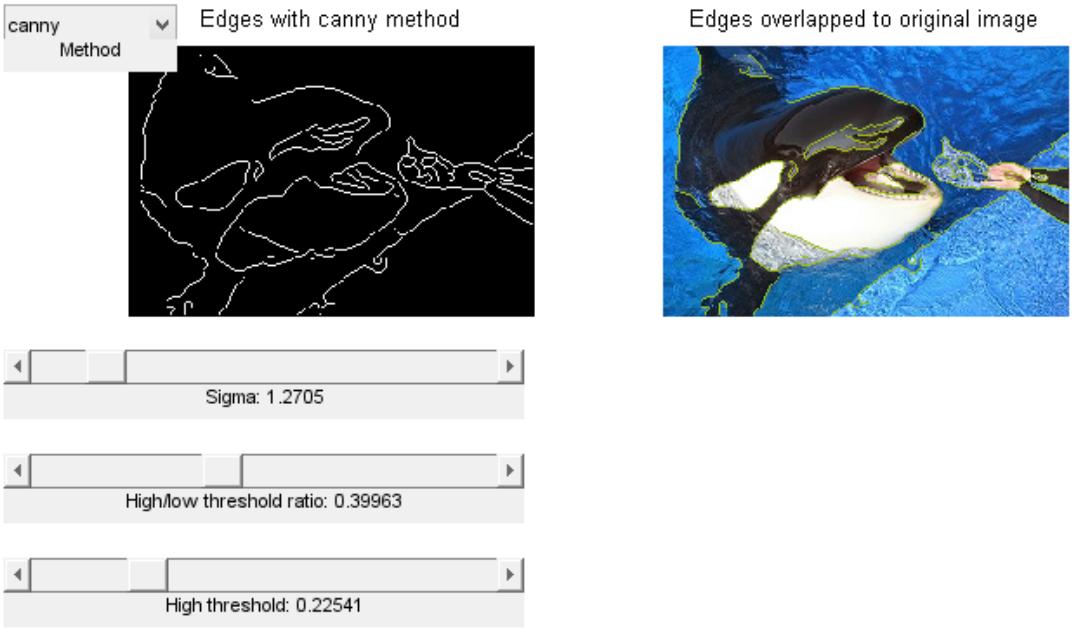

Figure 12: Selected parameters for extracting the edges from `doulphin.jpg` logo with Canny

its parameters while the video is being shown. Some examples are shown in figures 13, 14, 15 and 15.

For this exercise we have decided to load frame after frame, process it and then show the image with the overlapped edges. The idea behind this is to bound the usage of memory. Therefore the relevance of the efficiency of the detectors came into play. We have added to our method the functionality of displaying the mean time between consecutive frames. For very slow detectors (Canny) the reproduction of the video was to slow (the time between frames was greater than 160ms in a 1.8GHz computer) while with less sophisticated detectors like Sobel the execution was considerably more agile (time between frames lower than 70ms). This has to be taken into account in real-time applications (e.g. in robotics).

For illustration purposes, we have selected three frames from the video and extracted their edges using all the detectors but the simplest one. The comparisons are shown in figures 17, 18 and 19.

Again, which is the best detector depends upon the application. If we want it for a real-time, we would probably prefer Sobel, Prewitt or Roberts (maybe even LoG or Zero-cross). If we can afford to spend some time to process the frames beforehand, Canny would yield very good results. For the particular case of this video, where all the frames have the same resolution, similar amount of noise and similar color distributions and shapes, it could be a good idea to tweak the parameters for a particular method. However, we have found that letting the detector choose the thresholds automatically is a good compromise between adaptability and quality. This is specially true in videos where there are abrupt changes of scenes among different sections.

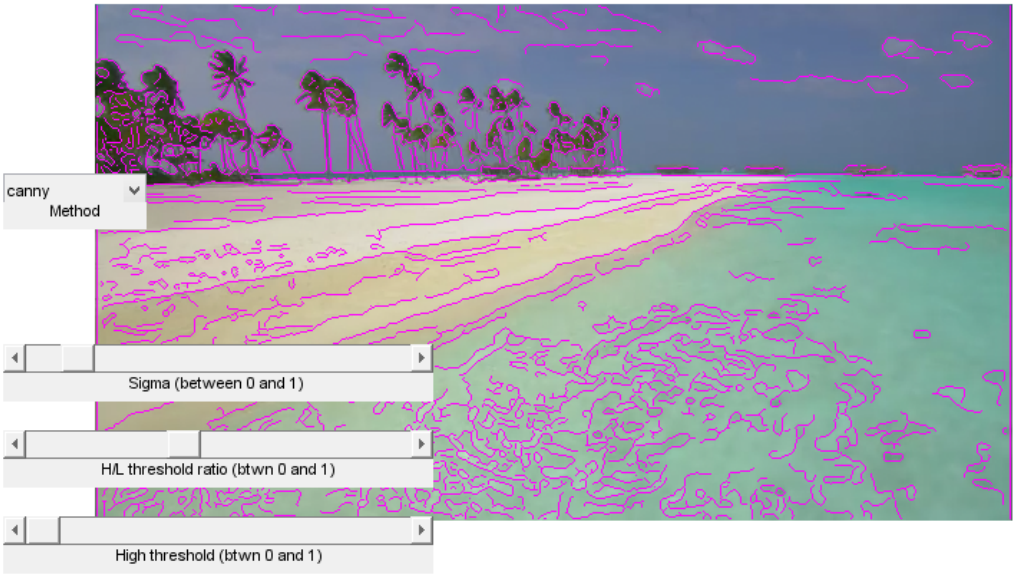

Figure 13: Beach with overlapped edges (Canny)

Figure 14: Beach with overlapped edges (Sobel)

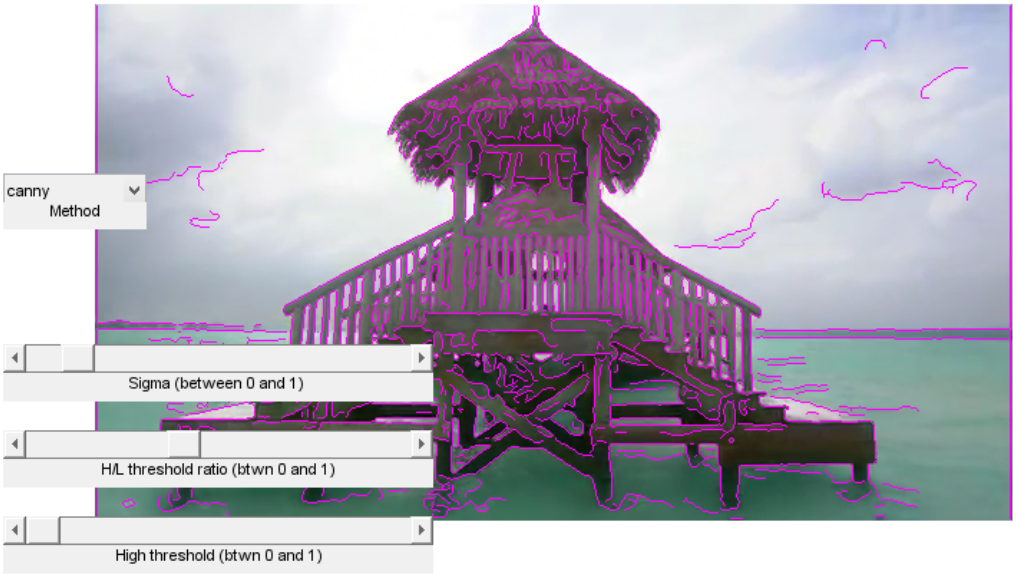

Figure 15: Cottage with overlapped edges (Canny)

Figure 16: Cottage with overlapped edges (Sobel)

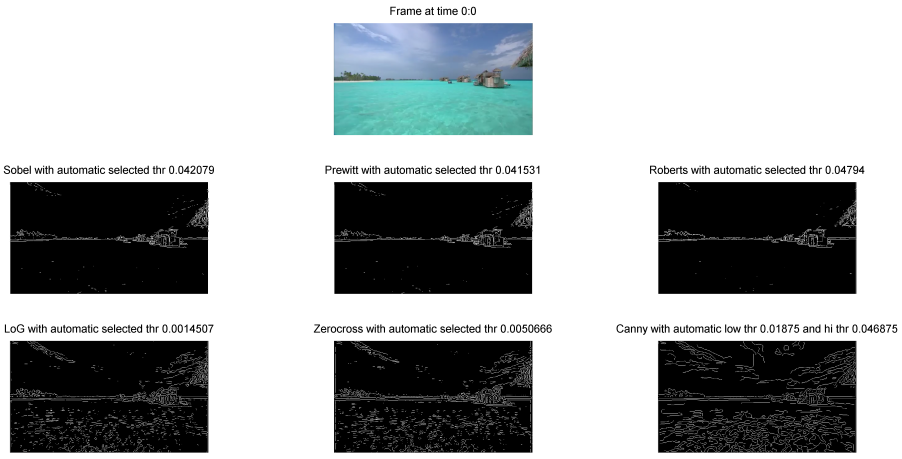

Figure 17: Comparison of edge detectors at time 0:0

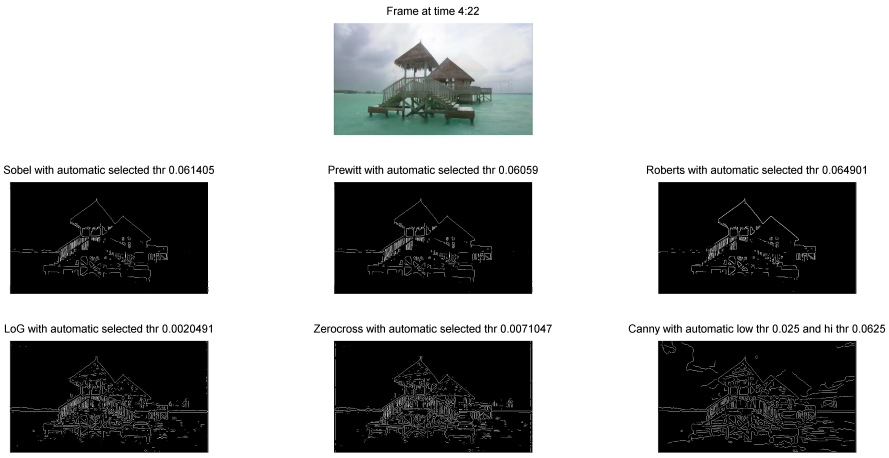

Figure 18: Comparison of edge detectors at time 4:22

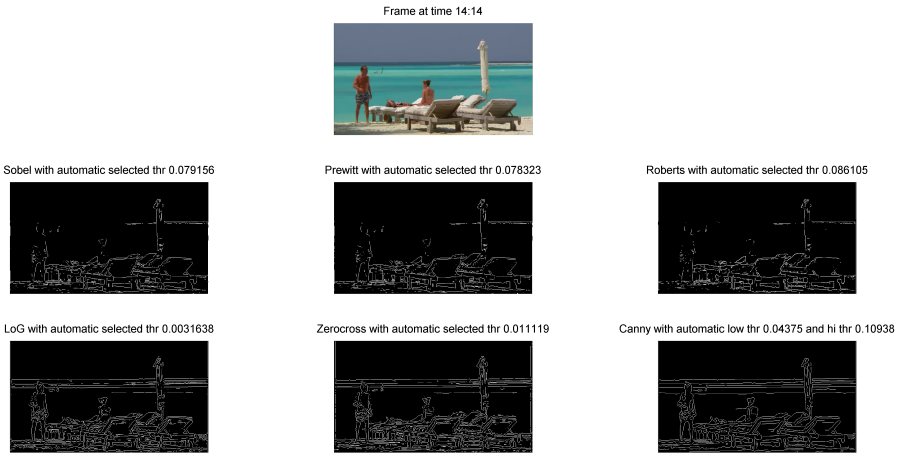

Figure 19: Comparison of edge detectors at time 14:14

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.