Implementing Sobel & Canny Edge Detection Algorithms

And comparing the results with built-in functions of Matlab

Ariyan Zarei ▸ 2/23/2017

Abstract

This is the report for the second project of the Image Processing Course (2nd Semester of the 95-96) by Dr. Alireza Tavakoli. As it is mentioned in the title of the report, we implemented sobel and canny edge detection algorithms in Matlab and compared the results acquired from them with the results from built-in functions in Matlab. The deliverables of the project consist of 8 .m files, each representing a function.

Implementing Sobel & Canny Edge Detection Algorithms

And comparing the results with built-in functions of Matlab

Introduction

In this project we implemented the sobel and canny edge detection algorithms from scratch and we compared the performances of our functions with built-in functions in Matlab. More precisely, we implemented 3 functions for each sobel and canny filter. The first function (MyCannyEdgeDetector / MySobelEdgeDetector) is the one we implemented from scratch. We implemented the convolution operation (MyConv function) and we implemented the whole algorithm for the canny/sobel method completely for this function. The second function (MatlabCannyEdgeDetector / MatlabSobelEdgeDetector) is the one we implemented the whole canny/sobel algorithm in it but we used "conv2" function instead of using MyConv function for convolution. But the third function (MatlabEdgeFunctionCanny / MatlabEdgeFunctionSobel) is just the encapsulation of the Matlab "edge" function which interestingly finds the edge for its input image.

How to Test the project

In order to test the project, first you need to read an image into your Matlab workspace by using "imread" function. This image can be either grayscale or RGB. Now to find the edges of this image with all the 6 possible functions all at once, you need to call the "Comparing" function and give this image as the only input for this function. But before that you have to change the active directory of Matlab to the directory in which the functions .m files exist. The result of calling the "Comparing" function is a figure in which there are 7 images. The image on top of the figure is the input image and the other images are the results of using different methods implemented earlier. You can see 3 different figures at the end of this report.

Functions

In the following sections you can see the necessary notes on each of the functions.

MyConv Function

Input: Image, Kernel, ShowResult
output: Result

This function computes the valid convolution of the Image matrix and Kernel matrix. Image can be either grayscale or RGB. The parameter ShowResult, indicates the user option whether to show the Result image at the end of the function or not. The function contains comments in order to shed light on the process of computing the convolution.

Comparing

Input: Image
Output: nothing.

This function represent a figure in which exist 6 images for each function of edge detection.

MySobelEdgeDetector

Input: Image
Output: Result

This function takes an image and produces edges using the sobel filter. The process consist of 4 steps. First the vertical and horizontal sobel filter is computed and using the below formula the image after performing both filters, is produced.

$$\sqrt{{H_{sobel}}^2 + {V_{sobel}}^2}$$

Then the grayscale output is converted to binary image using "im2bw" function and Otsu threshold and as the last step, using "bwmorph" the skeleton of the image is generated in order to represent the edges as narrow as possible.

MatlabSobelEdgeDetector

This function acts the same as the previous function, except for the convolution. In this function convolution is calculated using "conv2" function.

MatlabEdgeFunctionSobel

Input and output for this function is the same as the others. But in this function we just used "edge" function of the Matlab which simply returns an image containing only the edges of the main image. We do not perform skeletonization on the output image in this function.

MyCannyEdgeDetector

Before getting into the description of this special function, we need to mention the steps used in the canny filter. Actually canny filter is a multi-step algorithm with the below steps. We implemented all of them in this and the next function.

1. Performing Gaussian filter in order to reduce the noise. We used "imfilter" and a 5x5 Gaussian kernel.
2. Finding the basic edges using a basic filter like sobel, which we actually used the sobel filter to find the derivatives in each horizontal and vertical directions.
3. Performing non-maximum suppression in order to thin the edges. This is done by first finding the direction of the edge for each pixel and then finding the Pa, Pb and P as below and at the end removing the pixel which are not match the condition Pa<P<Pb.

   ```
   Theta = Atan2(H,V);
   Pa = TotalSobel(i_1,j_1);
   Pb = TotalSobel(i_2,j_2);
   P = TotalSobel(i,j);
   ```

   Which i and j indices are the desired indices with respect to the Theta.

4. At the final step we perform a two-step threshold using t1 and t2 as below:
   - If a pixel is below the t1 threshold, we remove it from the result (by changing its intensity to 0).
   - If a pixel intensity is between t1 and t2, we only keep it in the result, if there exists a path of other similar pixels from this pixel to a pixel with intensity more than t2.
   - We keep other pixels which have intensity more than t2.

   This algorithm can be implemented using a recursive approach which makes the canny filter very slow with respect to the execution time.

   We should mention that we considered t1 = 20 and t2 = 80 in our function based on the desired results.

MatlabCannyEdgeDetector

This function has the same implementation as the previous function but only with a difference in convolution part like the other two sobel method had. In this function we used "conv2" function of the Matlab.

It is necessary to mention that we did performed skeletonization in this and previous function.

MatlabEdgeFunctionCanny

Here in this function in order to find the edges with respect to the canny filter, like the MatlabEdgeFunctionSobel function, we used the "edge" function of the Matlab to find the edges of the input image. Again, we didn't performed other skeletonization and binarisation in this function.

Results and Conclusion

As you can see the results below, it seems that for images with high details, as for image 1 and 2, the "edge" function of the Matlab using canny filter, has the best result and for the images with low details like image 3, sobel filter implemented using "conv2" of Matlab, produces the most accurate result.

Image 1

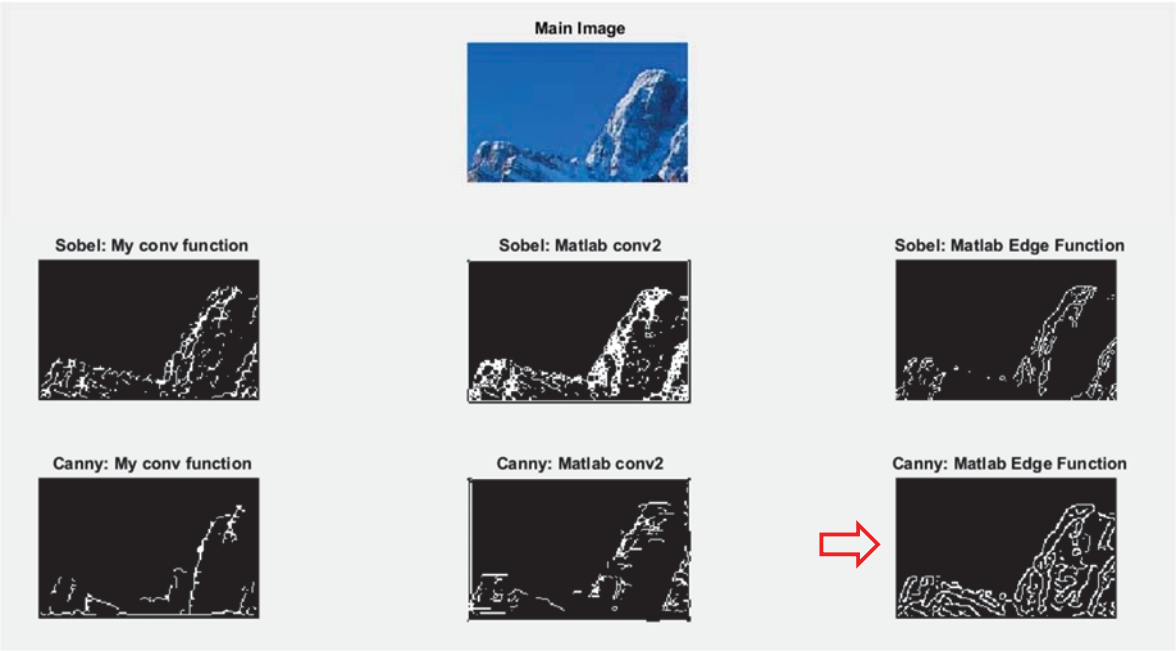


Image 2

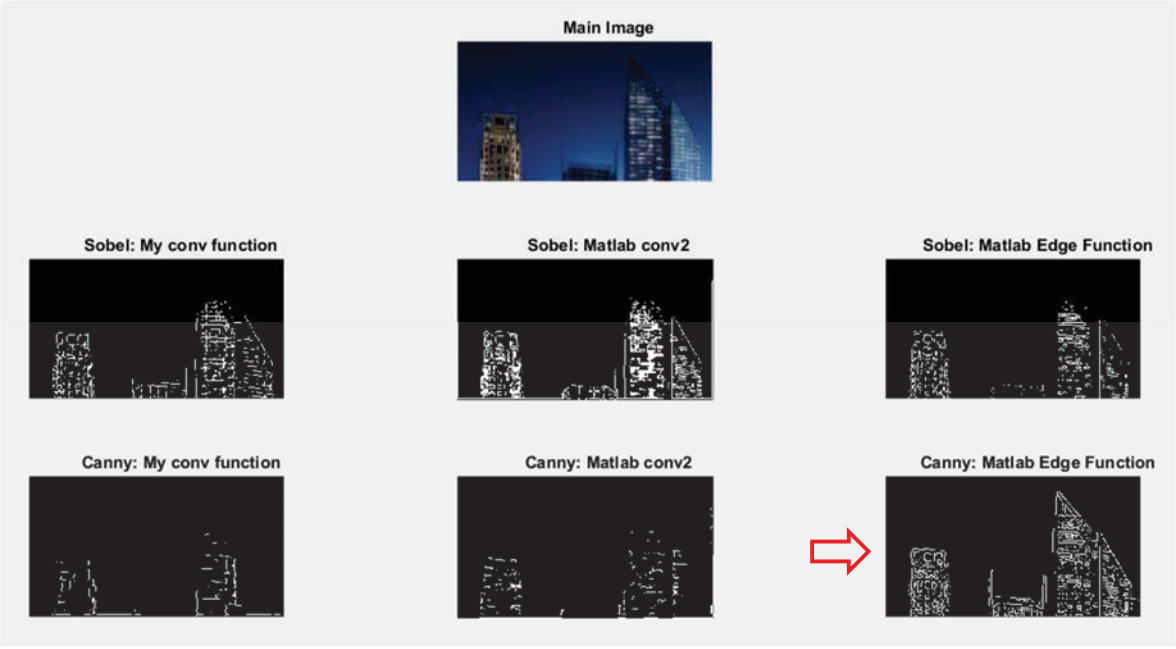
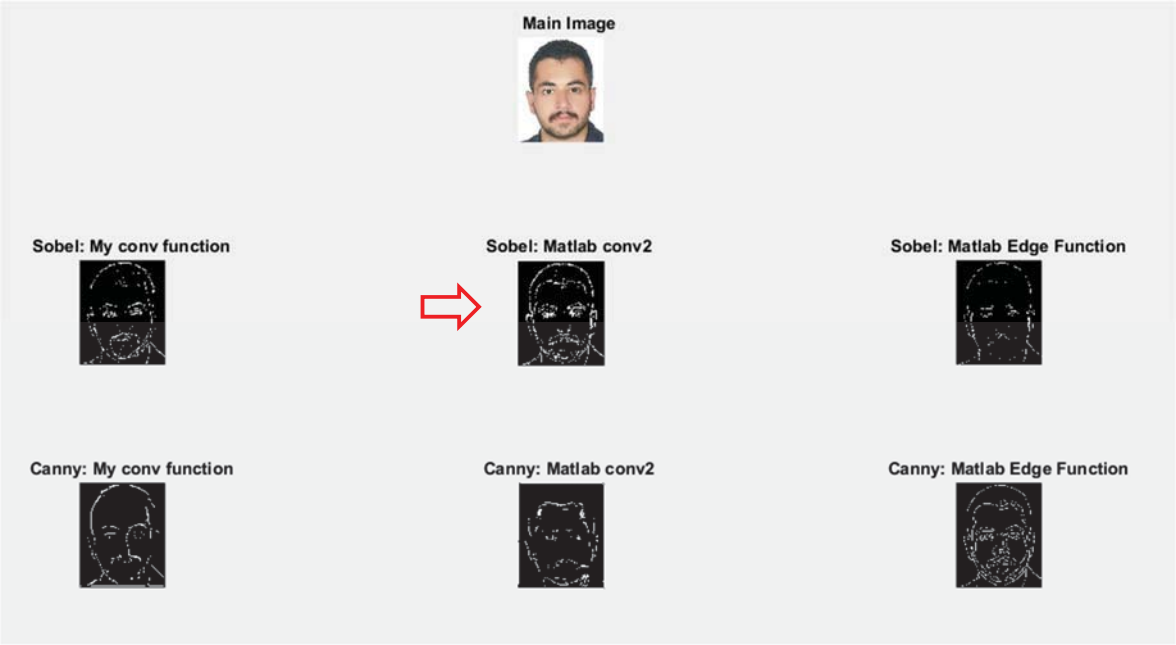


Image 3


Main Image
Sobel: My conv function
Sobel: Matlab conv2
Sobel: Matlab Edge Function
Canny: My conv function
Canny: Matlab conv2
Canny: Matlab Edge Function