**Sobel Derivatives**

**Prev Tutorial:** [**Adding borders to your images**](https://docs.opencv.org/3.4/dc/da3/tutorial_copyMakeBorder.html)

**Next Tutorial:** [**Laplace Operator**](https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html)

**Goal**

In this tutorial you will learn how to:

* Use the OpenCV function [**Sobel()**](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gacea54f142e81b6758cb6f375ce782c8d) to calculate the derivatives from an image.
* Use the OpenCV function [**Scharr()**](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaa13106761eedf14798f37aa2d60404c9) to calculate a more accurate derivative for a kernel of size 3⋅3

**Theory**

**Note**

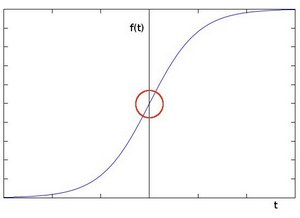
The explanation below belongs to the book **Learning OpenCV** by Bradski and Kaehler.

1. In the last two tutorials we have seen applicative examples of convolutions. One of the most important convolutions is the computation of derivatives in an image (or an approximation to them).
2. Why may be important the calculus of the derivatives in an image? Let's imagine we want to detect the *edges* present in the image. For instance:

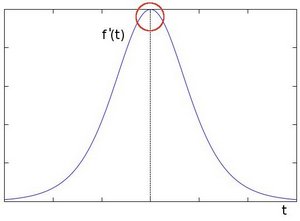


You can easily notice that in an *edge*, the pixel intensity *changes* in a notorious way. A good way to express *changes* is by using *derivatives*. A high change in gradient indicates a major change in the image.

1. To be more graphical, let's assume we have a 1D-image. An edge is shown by the "jump" in intensity in the plot below:



1. The edge "jump" can be seen more easily if we take the first derivative (actually, here appears as a maximum)



1. So, from the explanation above, we can deduce that a method to detect edges in an image can be performed by locating pixel locations where the gradient is higher than its neighbors (or to generalize, higher than a threshold).
2. More detailed explanation, please refer to **Learning OpenCV** by Bradski and Kaehler

**Sobel Operator**

1. The Sobel Operator is a discrete differentiation operator. It computes an approximation of the gradient of an image intensity function.
2. The Sobel Operator combines Gaussian smoothing and differentiation.

**Formulation**

Assuming that the image to be operated is I:

1. We calculate two derivatives:
   1. **Horizontal changes**: This is computed by convolving I with a kernel Gx with odd size. For example for a kernel size of 3, Gx would be computed as:

Gx=⎡⎣⎢−1−2−1000+1+2+1⎤⎦⎥∗I

* 1. **Vertical changes**: This is computed by convolving I with a kernel Gy with odd size. For example for a kernel size of 3, Gy would be computed as:

Gy=⎡⎣⎢−10+1−20+2−10+1⎤⎦⎥∗I

1. At each point of the image we calculate an approximation of the *gradient* in that point by combining both results above:

G=G2x+G2y−−−−−−−√

Although sometimes the following simpler equation is used:

G=|Gx|+|Gy|

**Note**

When the size of the kernel is 3, the Sobel kernel shown above may produce noticeable inaccuracies (after all, Sobel is only an approximation of the derivative). OpenCV addresses this inaccuracy for kernels of size 3 by using the [**Scharr()**](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaa13106761eedf14798f37aa2d60404c9) function. This is as fast but more accurate than the standard Sobel function. It implements the following kernels:

Gx=⎡⎣⎢−3−10−3000+3+10+3⎤⎦⎥

Gy=⎡⎣⎢−30+3−100+10−30+3⎤⎦⎥

You can check out more information of this function in the OpenCV reference - [**Scharr()**](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaa13106761eedf14798f37aa2d60404c9) . Also, in the sample code below, you will notice that above the code for [**Sobel()**](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gacea54f142e81b6758cb6f375ce782c8d) function there is also code for the [**Scharr()**](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaa13106761eedf14798f37aa2d60404c9) function commented. Uncommenting it (and obviously commenting the Sobel stuff) should give you an idea of how this function works.

**Laplace Operator**

**Prev Tutorial:** [**Sobel Derivatives**](https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html)

**Next Tutorial:** [**Canny Edge Detector**](https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html)

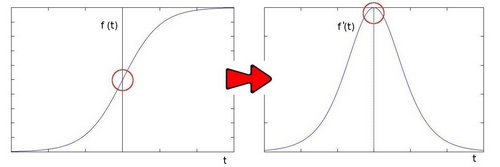
**Goal**

In this tutorial you will learn how to:

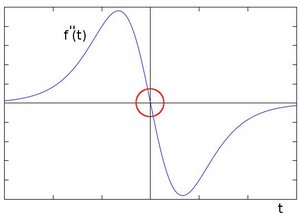
* Use the OpenCV function [**Laplacian()**](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gad78703e4c8fe703d479c1860d76429e6) to implement a discrete analog of the *Laplacian operator*.

**Theory**

1. In the previous tutorial we learned how to use the *Sobel Operator*. It was based on the fact that in the edge area, the pixel intensity shows a "jump" or a high variation of intensity. Getting the first derivative of the intensity, we observed that an edge is characterized by a maximum, as it can be seen in the figure:



1. And...what happens if we take the second derivative?



You can observe that the second derivative is zero! So, we can also use this criterion to attempt to detect edges in an image. However, note that zeros will not only appear in edges (they can actually appear in other meaningless locations); this can be solved by applying filtering where needed.

**Laplacian Operator**

1. From the explanation above, we deduce that the second derivative can be used to *detect edges*. Since images are "\*2D\*", we would need to take the derivative in both dimensions. Here, the Laplacian operator comes handy.
2. The *Laplacian operator* is defined by:

Laplace(f)=∂2f∂x2+∂2f∂y2

1. The Laplacian operator is implemented in OpenCV by the function [**Laplacian()**](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gad78703e4c8fe703d479c1860d76429e6) . In fact, since the Laplacian uses the gradient of images, it calls internally the *Sobel* operator to perform its computation.

**Code  
  
C++JavaPython**

1. **What does this program do?**
   * Loads an image
   * Remove noise by applying a Gaussian blur and then convert the original image to grayscale
   * Applies a Laplacian operator to the grayscale image and stores the output image
   * Display the result in a window
2. The tutorial code's is shown lines below. You can also download it from [here](https://raw.githubusercontent.com/opencv/opencv/3.4/samples/python/tutorial_code/ImgTrans/LaPlace/laplace_demo.py)

"""

@file laplace\_demo.py

@brief Sample code showing how to detect edges using the Laplace operator

"""

import sys

import cv2 as cv

def main(argv):

# [variables]

# Declare the variables we are going to use

ddepth = cv.CV\_16S

kernel\_size = 3

window\_name = "Laplace Demo"

# [variables]

# [load]

imageName = argv[0] if len(argv) > 0 else 'lena.jpg'

src = [cv.imread](https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56)([cv.samples.findFile](https://docs.opencv.org/3.4/d6/dba/group__core__utils__samples.html#ga3a33b00033b46c698ff6340d95569c13)(imageName), cv.IMREAD\_COLOR) # Load an image

# Check if image is loaded fine

if src is None:

print ('Error opening image')

print ('Program Arguments: [image\_name -- default lena.jpg]')

return -1

# [load]

# [reduce\_noise]

# Remove noise by blurring with a Gaussian filter

src = [cv.GaussianBlur](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1)(src, (3, 3), 0)

# [reduce\_noise]

# [convert\_to\_gray]

# Convert the image to grayscale

src\_gray = [cv.cvtColor](https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab)(src, cv.COLOR\_BGR2GRAY)

# [convert\_to\_gray]

# Create Window

[cv.namedWindow](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#ga5afdf8410934fd099df85c75b2e0888b)(window\_name, cv.WINDOW\_AUTOSIZE)

# [laplacian]

# Apply Laplace function

dst = [cv.Laplacian](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gad78703e4c8fe703d479c1860d76429e6)(src\_gray, ddepth, ksize=kernel\_size)

# [laplacian]

# [convert]

# converting back to uint8

abs\_dst = [cv.convertScaleAbs](https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga3460e9c9f37b563ab9dd550c4d8c4e7d)(dst)

# [convert]

# [display]

[cv.imshow](https://docs.opencv.org/3.4/df/d24/group__highgui__opengl.html#gaae7e90aa3415c68dba22a5ff2cefc25d)(window\_name, abs\_dst)

[cv.waitKey](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#ga5628525ad33f52eab17feebcfba38bd7)(0)

# [display]

return 0

if \_\_name\_\_ == "\_\_main\_\_":

main(sys.argv[1:])

**Explanation  
  
C++JavaPython**

**Declare variables**

# Declare the variables we are going to use

ddepth = cv.CV\_16S

kernel\_size = 3

window\_name = "Laplace Demo"

**Load source image**

imageName = argv[0] if len(argv) > 0 else 'lena.jpg'

src = [cv.imread](https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56)([cv.samples.findFile](https://docs.opencv.org/3.4/d6/dba/group__core__utils__samples.html#ga3a33b00033b46c698ff6340d95569c13)(imageName), cv.IMREAD\_COLOR) # Load an image

# Check if image is loaded fine

if src is None:

print ('Error opening image')

print ('Program Arguments: [image\_name -- default lena.jpg]')

return -1

**Reduce noise**

# Remove noise by blurring with a Gaussian filter

src = [cv.GaussianBlur](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1)(src, (3, 3), 0)

**Grayscale**

# Convert the image to grayscale

src\_gray = [cv.cvtColor](https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab)(src, cv.COLOR\_BGR2GRAY)

**Laplacian operator**

# Apply Laplace function

dst = [cv.Laplacian](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gad78703e4c8fe703d479c1860d76429e6)(src\_gray, ddepth, ksize=kernel\_size)

* The arguments are:
  + *src\_gray*: The input image.
  + *dst*: Destination (output) image
  + *ddepth*: Depth of the destination image. Since our input is *CV\_8U* we define *ddepth* = *CV\_16S* to avoid overflow
  + *kernel\_size*: The kernel size of the Sobel operator to be applied internally. We use 3 in this example.
  + *scale*, *delta* and *BORDER\_DEFAULT*: We leave them as default values.

**Convert output to a *CV\_8U* image**

# converting back to uint8

abs\_dst = [cv.convertScaleAbs](https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga3460e9c9f37b563ab9dd550c4d8c4e7d)(dst)

**Display the result**

[cv.imshow](https://docs.opencv.org/3.4/df/d24/group__highgui__opengl.html#gaae7e90aa3415c68dba22a5ff2cefc25d)(window\_name, abs\_dst)

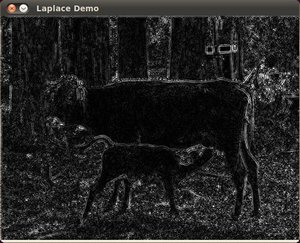
[cv.waitKey](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#ga5628525ad33f52eab17feebcfba38bd7)(0)

**Results**

1. After compiling the code above, we can run it giving as argument the path to an image. For example, using as an input:



1. We obtain the following result. Notice how the trees and the silhouette of the cow are approximately well defined (except in areas in which the intensity are very similar, i.e. around the cow's head). Also, note that the roof of the house behind the trees (right side) is notoriously marked. This is due to the fact that the contrast is higher in that region.



**Canny Edge Detector**

**Prev Tutorial:** [**Laplace Operator**](https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html)

**Next Tutorial:** [**Hough Line Transform**](https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html)

**Goal**

In this tutorial you will learn how to:

* Use the OpenCV function [**cv::Canny**](https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de) to implement the Canny Edge Detector.

**Theory**

The *Canny Edge detector* [**[34]**](https://docs.opencv.org/3.4/d0/de3/citelist.html#CITEREF_Canny86) was developed by John F. Canny in 1986. Also known to many as the *optimal detector*, the Canny algorithm aims to satisfy three main criteria:

* **Low error rate:** Meaning a good detection of only existent edges.
* **Good localization:** The distance between edge pixels detected and real edge pixels have to be minimized.
* **Minimal response:** Only one detector response per edge.

**Steps**

1. Filter out any noise. The Gaussian filter is used for this purpose. An example of a Gaussian kernel of size=5 that might be used is shown below:

K=1159⎡⎣⎢⎢⎢⎢⎢⎢245424912945121512549129424542⎤⎦⎥⎥⎥⎥⎥⎥

1. Find the intensity gradient of the image. For this, we follow a procedure analogous to Sobel:
   1. Apply a pair of convolution masks (in x and y directions:

Gx=⎡⎣⎢−1−2−1000+1+2+1⎤⎦⎥

Gy=⎡⎣⎢−10+1−20+2−10+1⎤⎦⎥

* 1. Find the gradient strength and direction with:

G=G2x+G2y−−−−−−−√θ=arctan(GyGx)

The direction is rounded to one of four possible angles (namely 0, 45, 90 or 135)

1. *Non-maximum* suppression is applied. This removes pixels that are not considered to be part of an edge. Hence, only thin lines (candidate edges) will remain.
2. *Hysteresis*: The final step. Canny does use two thresholds (upper and lower):
   1. If a pixel gradient is higher than the *upper* threshold, the pixel is accepted as an edge
   2. If a pixel gradient value is below the *lower* threshold, then it is rejected.
   3. If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the *upper* threshold.

Canny recommended a *upper*:*lower* ratio between 2:1 and 3:1.

1. For more details, you can always consult your favorite Computer Vision book.

**Code  
  
C++JavaPython**

* The tutorial code's is shown lines below. You can also download it from [here](https://github.com/opencv/opencv/tree/3.4/samples/cpp/tutorial_code/ImgTrans/CannyDetector_Demo.cpp)

#include "[opencv2/imgproc.hpp](https://docs.opencv.org/3.4/dd/d46/imgproc_8hpp.html)"

#include "[opencv2/highgui.hpp](https://docs.opencv.org/3.4/d4/dd5/highgui_8hpp.html)"

#include <iostream>

using namespace [cv](https://docs.opencv.org/3.4/d2/d75/namespacecv.html);

[Mat](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html) src, src\_gray;

[Mat](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html) dst, detected\_edges;

int lowThreshold = 0;

const int max\_lowThreshold = 100;

const int ratio = 3;

const int kernel\_size = 3;

const char\* window\_name = "Edge Map";

static void CannyThreshold(int, void\*)

{

[blur](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#ga8c45db9afe636703801b0b2e440fce37)( src\_gray, detected\_edges, [Size](https://docs.opencv.org/3.4/dc/d84/group__core__basic.html#ga346f563897249351a34549137c8532a0)(3,3) );

[Canny](https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de)( detected\_edges, detected\_edges, lowThreshold, lowThreshold\*ratio, kernel\_size );

dst = [Scalar::all](https://docs.opencv.org/3.4/d1/da0/classcv_1_1Scalar__.html#ac1509a4b8454fe7fe29db069e13a2e6f)(0);

src.[copyTo](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html#a33fd5d125b4c302b0c9aa86980791a77)( dst, detected\_edges);

[imshow](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563)( window\_name, dst );

}

int main( int argc, char\*\* argv )

{

[CommandLineParser](https://docs.opencv.org/3.4/d0/d2e/classcv_1_1CommandLineParser.html) parser( argc, argv, "{@input | fruits.jpg | input image}" );

src = [imread](https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56)( [samples::findFile](https://docs.opencv.org/3.4/d6/dba/group__core__utils__samples.html#ga3a33b00033b46c698ff6340d95569c13)( parser.get<[String](https://docs.opencv.org/3.4/d1/d8f/classcv_1_1String.html)>( "@input" ) ), [IMREAD\_COLOR](https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#gga61d9b0126a3e57d9277ac48327799c80af660544735200cbe942eea09232eb822) ); // Load an image

if( src.[empty](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html#abbec3525a852e77998aba034813fded4)() )

{

std::cout << "Could not open or find the image!\n" << std::endl;

std::cout << "Usage: " << argv[0] << " <Input image>" << std::endl;

return -1;

}

dst.[create](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html#a55ced2c8d844d683ea9a725c60037ad0)( src.[size](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html#a146f8e8dda07d1365a575ab83d9828d1)(), src.[type](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html#af2d2652e552d7de635988f18a84b53e5)() );

[cvtColor](https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab)( src, src\_gray, [COLOR\_BGR2GRAY](https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#gga4e0972be5de079fed4e3a10e24ef5ef0a353a4b8db9040165db4dacb5bcefb6ea) );

[namedWindow](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#ga5afdf8410934fd099df85c75b2e0888b)( window\_name, [WINDOW\_AUTOSIZE](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#ggabf7d2c5625bc59ac130287f925557ac3acf621ace7a54954cbac01df27e47228f) );

[createTrackbar](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#gaf78d2155d30b728fc413803745b67a9b)( "Min Threshold:", window\_name, &lowThreshold, max\_lowThreshold, CannyThreshold );

CannyThreshold(0, 0);

[waitKey](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#ga5628525ad33f52eab17feebcfba38bd7)(0);

return 0;

}

 **What does this program do?**

* Asks the user to enter a numerical value to set the lower threshold for our *Canny Edge Detector* (by means of a Trackbar).
* Applies the *Canny Detector* and generates a **mask** (bright lines representing the edges on a black background).
* Applies the mask obtained on the original image and display it in a window.

**Explanation (C++ code)**

1. Create some needed variables:

Mat src, src\_gray;

Mat dst, detected\_edges;

int lowThreshold = 0;

const int max\_lowThreshold = 100;

const int ratio = 3;

const int kernel\_size = 3;

const char\* window\_name = "Edge Map";

Note the following:

* 1. We establish a ratio of lower:upper threshold of 3:1 (with the variable *ratio*).
  2. We set the kernel size of 3 (for the Sobel operations to be performed internally by the Canny function).
  3. We set a maximum value for the lower Threshold of 100.

1. Loads the source image:

CommandLineParser parser( argc, argv, "{@input | fruits.jpg | input image}" );

src = [imread](https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56)( [samples::findFile](https://docs.opencv.org/3.4/d6/dba/group__core__utils__samples.html#ga3a33b00033b46c698ff6340d95569c13)( parser.get<String>( "@input" ) ), [IMREAD\_COLOR](https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#gga61d9b0126a3e57d9277ac48327799c80af660544735200cbe942eea09232eb822) ); // Load an image

if( src.empty() )

{

std::cout << "Could not open or find the image!\n" << std::endl;

std::cout << "Usage: " << argv[0] << " <Input image>" << std::endl;

return -1;

}

1. Create a matrix of the same type and size of *src* (to be *dst*):

dst.create( src.size(), src.type() );

1. Convert the image to grayscale (using the function [**cv::cvtColor**](https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab) ):

[cvtColor](https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab)( src, src\_gray, [COLOR\_BGR2GRAY](https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#gga4e0972be5de079fed4e3a10e24ef5ef0a353a4b8db9040165db4dacb5bcefb6ea) );

1. Create a window to display the results:

[namedWindow](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#ga5afdf8410934fd099df85c75b2e0888b)( window\_name, [WINDOW\_AUTOSIZE](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#ggabf7d2c5625bc59ac130287f925557ac3acf621ace7a54954cbac01df27e47228f) );

1. Create a Trackbar for the user to enter the lower threshold for our Canny detector:

[createTrackbar](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#gaf78d2155d30b728fc413803745b67a9b)( "Min Threshold:", window\_name, &lowThreshold, max\_lowThreshold, CannyThreshold );

Observe the following:

* 1. The variable to be controlled by the Trackbar is *lowThreshold* with a limit of *max\_lowThreshold* (which we set to 100 previously)
  2. Each time the Trackbar registers an action, the callback function *CannyThreshold* will be invoked.

1. Let's check the *CannyThreshold* function, step by step:
   1. First, we blur the image with a filter of kernel size 3:

[blur](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#ga8c45db9afe636703801b0b2e440fce37)( src\_gray, detected\_edges, [Size](https://docs.opencv.org/3.4/dc/d84/group__core__basic.html#ga346f563897249351a34549137c8532a0)(3,3) );

* 1. Second, we apply the OpenCV function [**cv::Canny**](https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de) :

[Canny](https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de)( detected\_edges, detected\_edges, lowThreshold, lowThreshold\*ratio, kernel\_size );

where the arguments are:

* + - *detected\_edges*: Source image, grayscale
    - *detected\_edges*: Output of the detector (can be the same as the input)
    - *lowThreshold*: The value entered by the user moving the Trackbar
    - *highThreshold*: Set in the program as three times the lower threshold (following Canny's recommendation)
    - *kernel\_size*: We defined it to be 3 (the size of the Sobel kernel to be used internally)

1. We fill a *dst* image with zeros (meaning the image is completely black).

dst = Scalar::all(0);

1. Finally, we will use the function [**cv::Mat::copyTo**](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html#a33fd5d125b4c302b0c9aa86980791a77) to map only the areas of the image that are identified as edges (on a black background). [**cv::Mat::copyTo**](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html#a33fd5d125b4c302b0c9aa86980791a77) copy the *src* image onto *dst*. However, it will only copy the pixels in the locations where they have non-zero values. Since the output of the Canny detector is the edge contours on a black background, the resulting *dst* will be black in all the area but the detected edges.

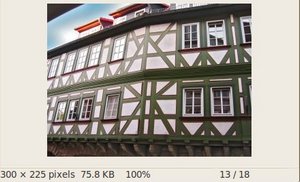
src.copyTo( dst, detected\_edges);

1. We display our result:

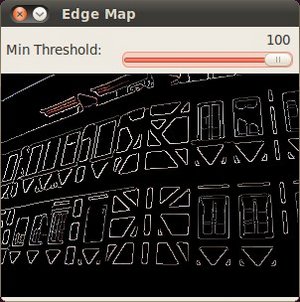
[imshow](https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563)( window\_name, dst );

**Result**

* After compiling the code above, we can run it giving as argument the path to an image. For example, using as an input the following image:



* Moving the slider, trying different threshold, we obtain the following result:



* Notice how the image is superposed to the black background on the edge regions.