**Problem statement:**

Implement canny edge detector algorithm from scratch using c++ to detect the edges is an Image. Neat Documentation is expected with code, Explanation , Input and Output Image.

**Explanation**:

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";

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 );

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 );

**Coding**:

#include<iostream>

#include "opencv2/opencv.hpp"

#include "opencv2/highgui.hpp"

using namespace cv;

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";

static void CannyThreshold(int, void\*)

{

blur( src\_gray, detected\_edges, Size(3,3) );

Canny( detected\_edges, detected\_edges, lowThreshold, lowThreshold\*ratio, kernel\_size );

dst = Scalar::all(0);

src.copyTo( dst, detected\_edges);

imshow( window\_name, dst );

}

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

{

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

src = imread( samples::findFile( parser.get<String>( "@input" ) ), IMREAD\_COLOR ); // 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;

}

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

cvtColor( src, src\_gray, COLOR\_BGR2GRAY );

namedWindow( window\_name, WINDOW\_AUTOSIZE );

createTrackbar( "Min Threshold:", window\_name, &lowThreshold, max\_lowThreshold, CannyThreshold );

CannyThreshold(0, 0);

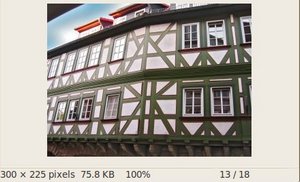
waitKey(0);

return 0;

}

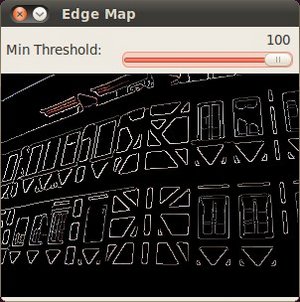
}

**Input Image:**



**Output Image:**

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



**Program Logic:**

* **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.

To detect the edges of the Image we can use the canny edge detection algorithm.

* Apply Guassian filter to smooth the image in order to remove the noise.
* Find the intensity gradients of the image.
* Apply non-maximum suppression to get rid of spurious response to edge detection.
* Apply double threshold to determine potential edges.
* Track edge by hysteresis, finally the detection by edges by suppressing all the other edges that are weak and not connected to strong edges.