**Assignment 1**

Makar Riabcev (2633590)

1. ***Image Filtering***

Code:

#include <stdio.h>

#include <opencv2/opencv.hpp>

#include <iostream>

using namespace cv;

using namespace std;

//our matrices

Mat kernel = (Mat\_<float>(3, 3) <<

0, -1, 0,

-1, 4, -1,

0, -1, 0);

Mat kernel2 = (Mat\_<float>(3, 3) <<

0, -1, 0,

-1, 8, -1,

0, -1, 0);

Mat kernelGrad = (Mat\_<float>(3, 3) <<

-1, 0, 1,

-1, 0, 1,

-1, 0, 1);

//function to apply kernel matrices to passed pictures

Mat filter(Mat inputKernel, Mat inputImage) {

Rect roi;

Mat frame;

int window\_size = 3;

Mat imageOutput = Mat::zeros(inputImage.size(), CV\_32F);

for (int y = 0; y < inputImage.rows - window\_size; ++y)

{

for (int x = 0; x < inputImage.cols - window\_size; ++x)

{

roi = Rect(x, y, window\_size, window\_size);

frame = inputImage(roi);

frame.convertTo(frame, CV\_32F);

frame = frame.mul(inputKernel);

float v = sum(frame)[0];

imageOutput.at<float>(y, x) = v;

}

}

imageOutput.convertTo(imageOutput, CV\_8U);

return imageOutput;

}

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

{

Mat image,imageResult,imageDif,imageGrad;

image = imread("C://edge detection.jpg", 0);

namedWindow("Original", WINDOW\_AUTOSIZE);

imshow("Original", image);

imageResult = filter(kernel, image);

namedWindow("Laplacian", CV\_WINDOW\_AUTOSIZE);

imshow("Laplacian", imageResult);

imageDif = filter(kernel2, image);

namedWindow("Laplacian2", WINDOW\_AUTOSIZE);

imshow("Laplacian2", imageDif);

imageGrad = filter(kernelGrad, image);

namedWindow("Gradient", WINDOW\_AUTOSIZE);

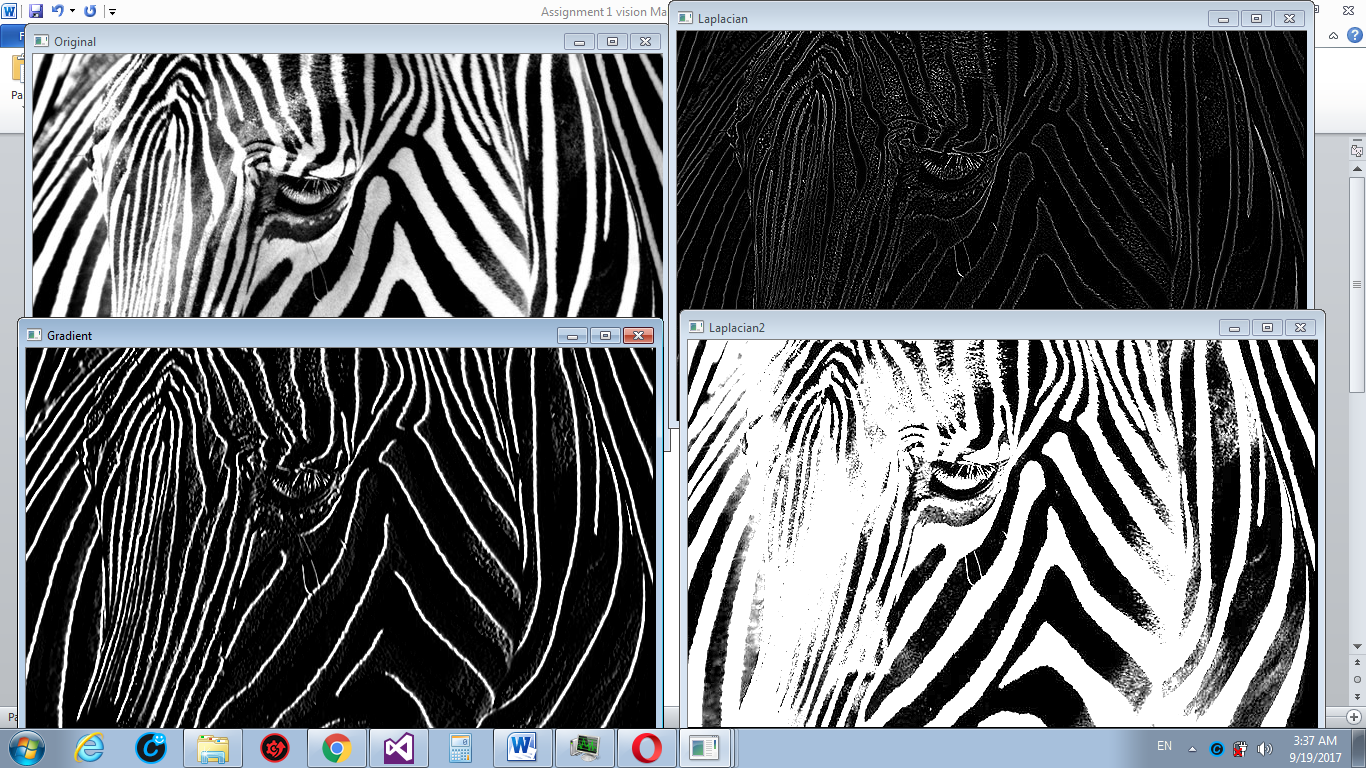
imshow("Gradient", imageGrad);

waitKey(0);

return 0;

}

Result:



Comments:

According to the results of Laplacian filter we can state what the higher anchor value of kernel the more filtered image looks like the original one.

1. ***Region Labelling Blob***

Code:

#include <iostream>

#include <vector>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

using namespace cv;

using namespace std;

void FindBlobs(const Mat &binary, vector < vector<Point2i> > &blobs);

int main()

{

Mat image = imread("C://blobs labelling.jpg", 0);

Mat output = Mat::zeros(image.size(), CV\_8UC3);

Mat binary;

vector < vector<Point2i > > blobs;

threshold(image, binary, 0.0, 1.0, cv::THRESH\_BINARY);

FindBlobs(binary, blobs);

for (size\_t i = 0; i < blobs.size(); i++) {

unsigned char r = 255 \* (rand() / (1.0 + RAND\_MAX));

unsigned char g = 255 \* (rand() / (1.0 + RAND\_MAX));

unsigned char b = 255 \* (rand() / (1.0 + RAND\_MAX));

for (size\_t j = 0; j < blobs[i].size(); j++) {

int x = blobs[i][j].x;

int y = blobs[i][j].y;

output.at<Vec3b>(y, x)[0] = b;

output.at<Vec3b>(y, x)[1] = g;

output.at<Vec3b>(y, x)[2] = r;

}

}

namedWindow("binary");

namedWindow("labelled");

imshow("binary", image);

imshow("labelled", output);

waitKey(0);

return 0;

}

void FindBlobs(const Mat &binary, vector < vector<Point2i> > &blobs)

{

blobs.clear();

// 0 - background

// 1 - unlabelled foreground

// 2+ - labelled foreground

Mat label\_image;

binary.convertTo(label\_image, CV\_32SC1);

int label\_count = 2; // start at 2 because 0,1 are used already

for (int y = 0; y < label\_image.rows; y++) {

int \*row = (int\*)label\_image.ptr(y);

for (int x = 0; x < label\_image.cols; x++) {

if (row[x] != 1) {

continue;

}

Rect rect;

floodFill(label\_image, Point(x, y), label\_count, &rect, 0, 0, 4);

vector <Point2i> blob;

for (int i = rect.y; i < (rect.y + rect.height); i++) {

int \*row2 = (int\*)label\_image.ptr(i);

for (int j = rect.x; j < (rect.x + rect.width); j++) {

if (row2[j] != label\_count) {

continue;

}

blob.push\_back(Point2i(j, i));

}

}

blobs.push\_back(blob);

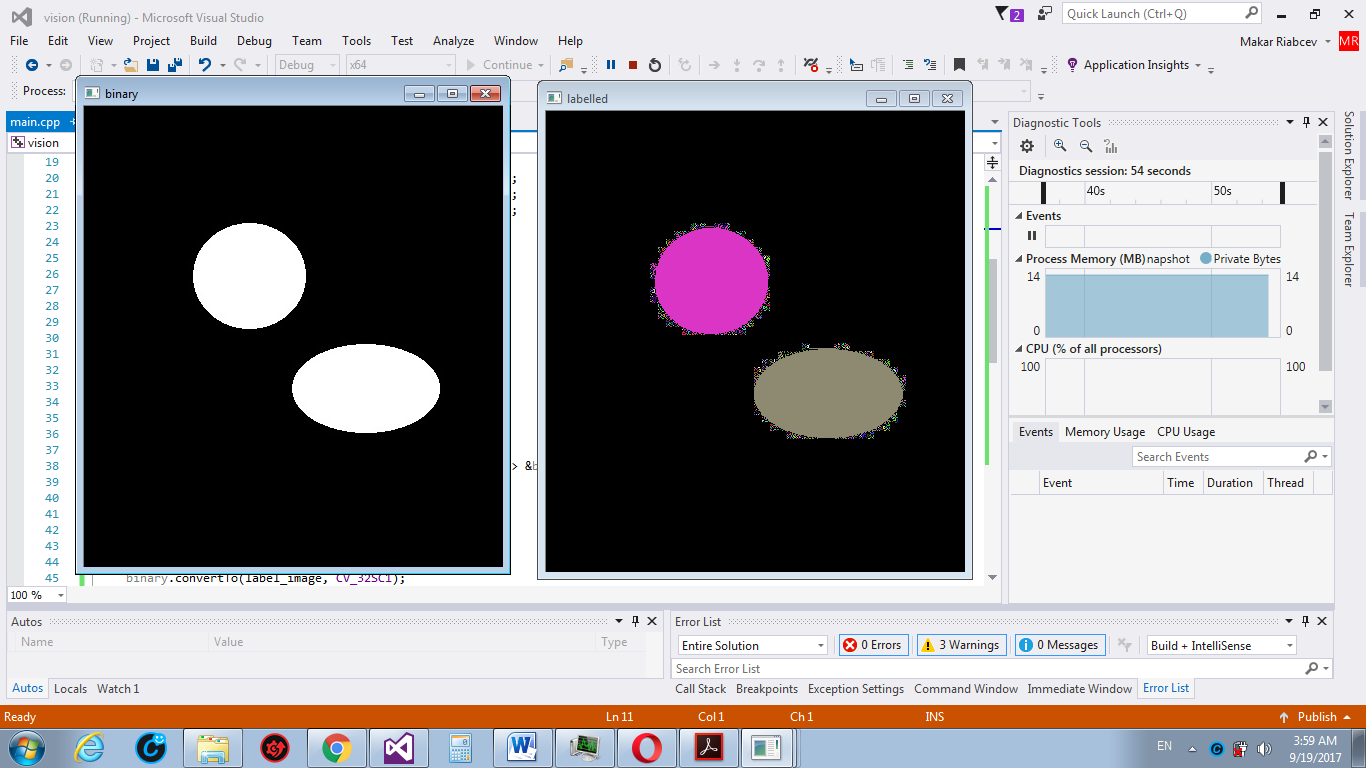
label\_count++;

}

}

}

Result:



Comments:

As we can see the application works correct. It labels the blobs with different colors.

Sources:

<https://stackoverflow.com/questions/11878281/image-sharpening-using-laplacian-filter>