**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

отчет

**по лабораторной работе №1**

**по дисциплине «Видеоаналитика и технологии обработки видеоданных в прикладных ТВ системах»**

Тема: **СЕГМЕНТАЦИЯ ДВИЖУЩИХСЯ ОБЪЕКТОВ НА ОСНОВЕ**

**ОЦЕНКИ ЭНЕРГИИ ДВИЖЕНИЯ**

Вариант

|  |  |  |
| --- | --- | --- |
| Студенты гр. 4105 |  | Суханов Р. А.  Гусев Д. В. |
| Преподаватель |  | Поздеев А. А. |

Санкт-Петербург

2019

**1 Цель работы.**

Разработать программу, реализующую алгоритм сегментации движущегося объекта с помощью оценки энергии движения. Исходными данными являются два кадра видеопоследовательности со статичным фоном, на которых присутствует перемещающийся объект.

**2 Алгоритм действий**

В программе должны быть реализованы следующие процедуры, с выводом результата на экран:

 загрузка пары кадров,

 получение препарата межкадровой разности,

 получение контурного препарата с помощью одного из алгоритмов,

указанных преподавателем: Собеля, разницы гауссиан, Кэнни,

**Робертса**.

 объединение препаратов по логическому «И»

 обработка сформированного изображения с помощью

морфологических фильтров (фильтрация и заливка),

 наложение строба на объект.

**3. Краткие теоретические сведения:**

При разработке алгоритмов сегментации объектов с учетом признака

движения необходимо учитывать характер движения. Область интереса может

обладать жестким (ригидным) и нежестким движением. Жесткое предполагает

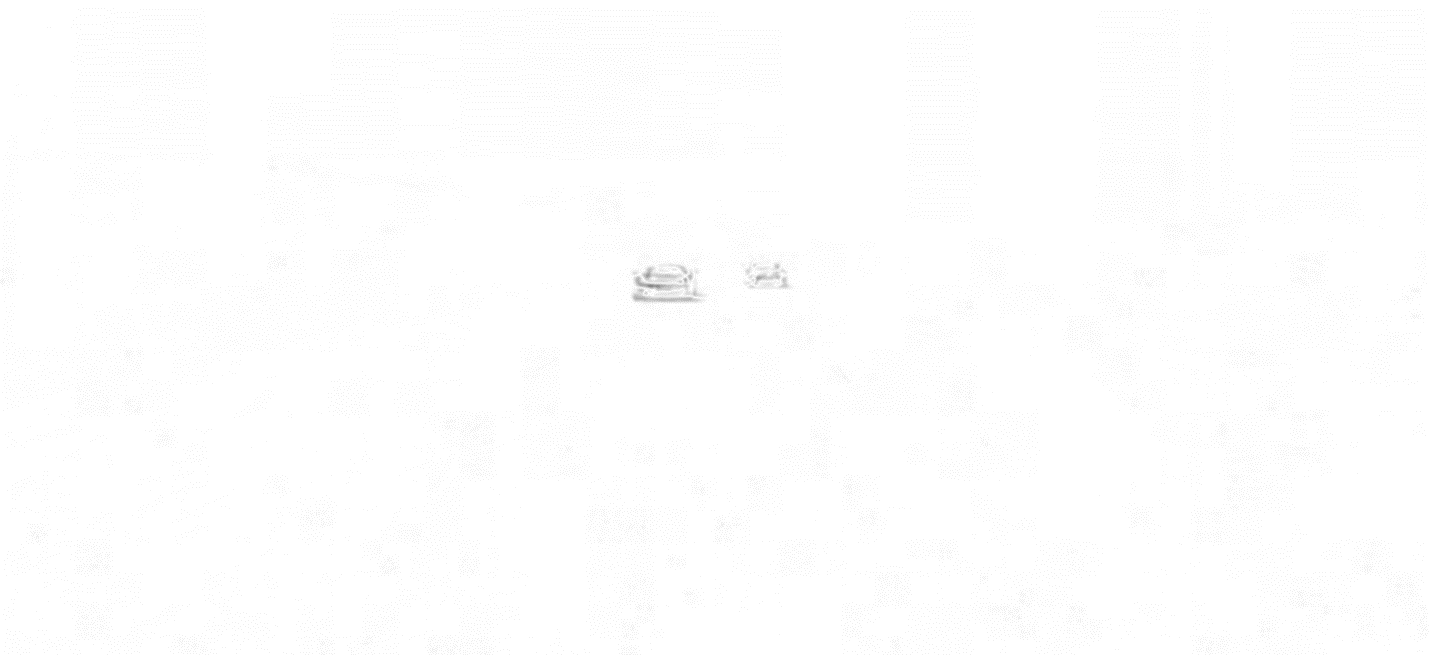
перемещение всех элементов объекта, в соответствии с основным направлением движения. Примерами являются движущийся автомобиль, летательный аппарат, корабль. Нежестким движением обладают, например, идущий человек, летящая птица.

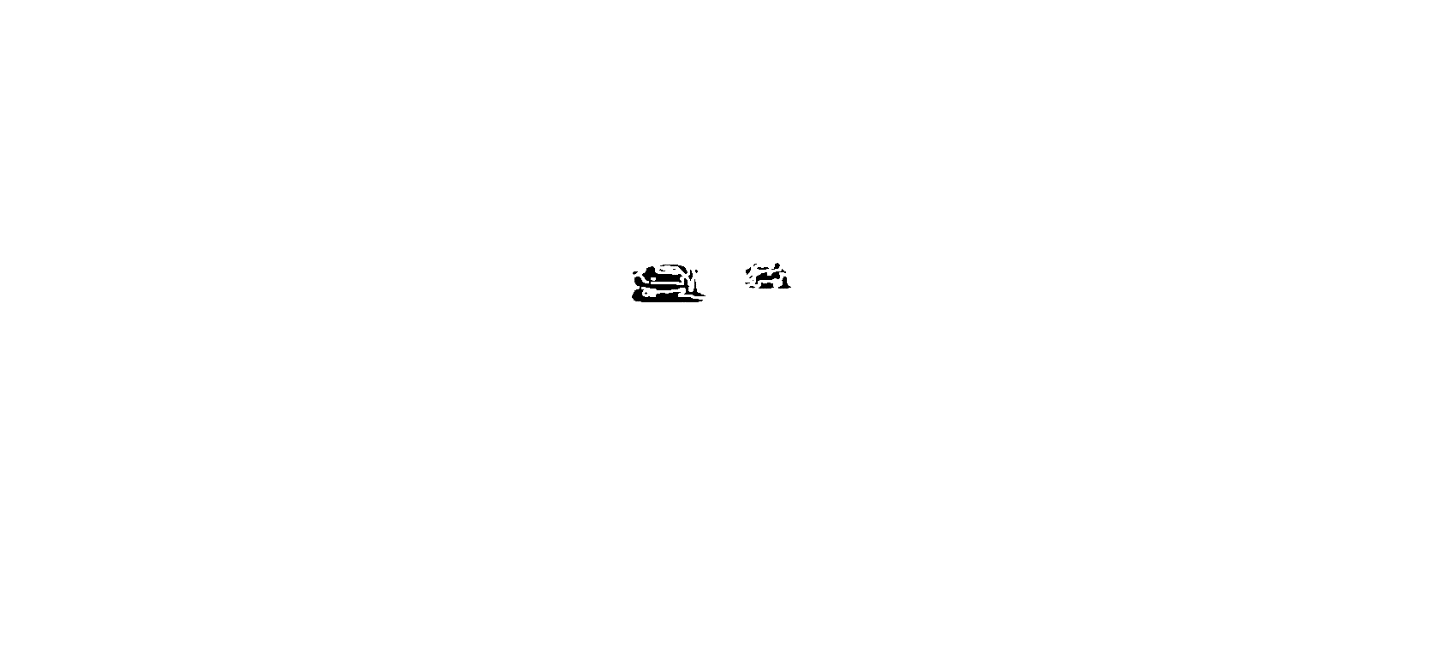
Достаточно простой и эффективный метод сегментации объектов на основе признака движения основан на оценке «энергии движения».

**4. Результаты работы:**

****

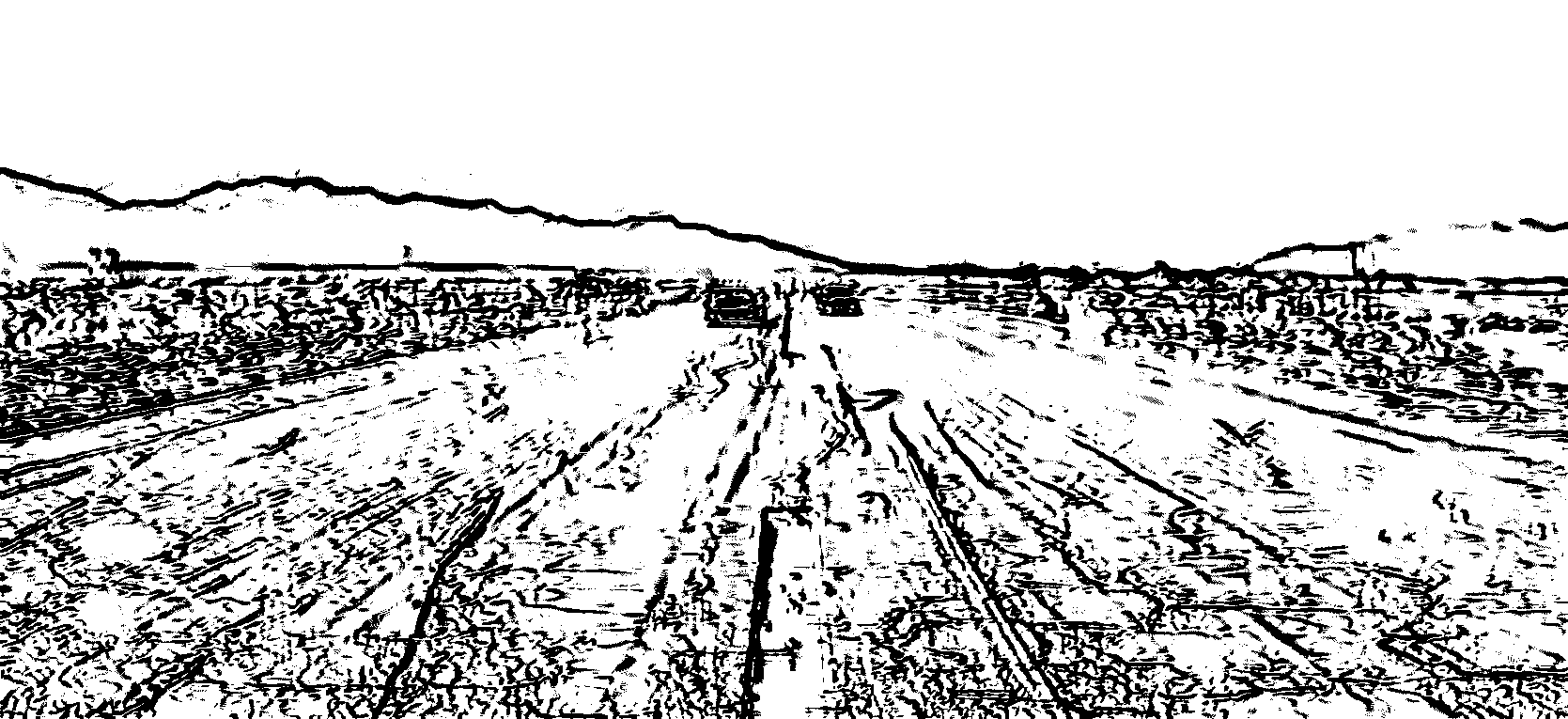
**Рисунок 1 – Преобразованные полутоновые изображения после медианной фильтрации**



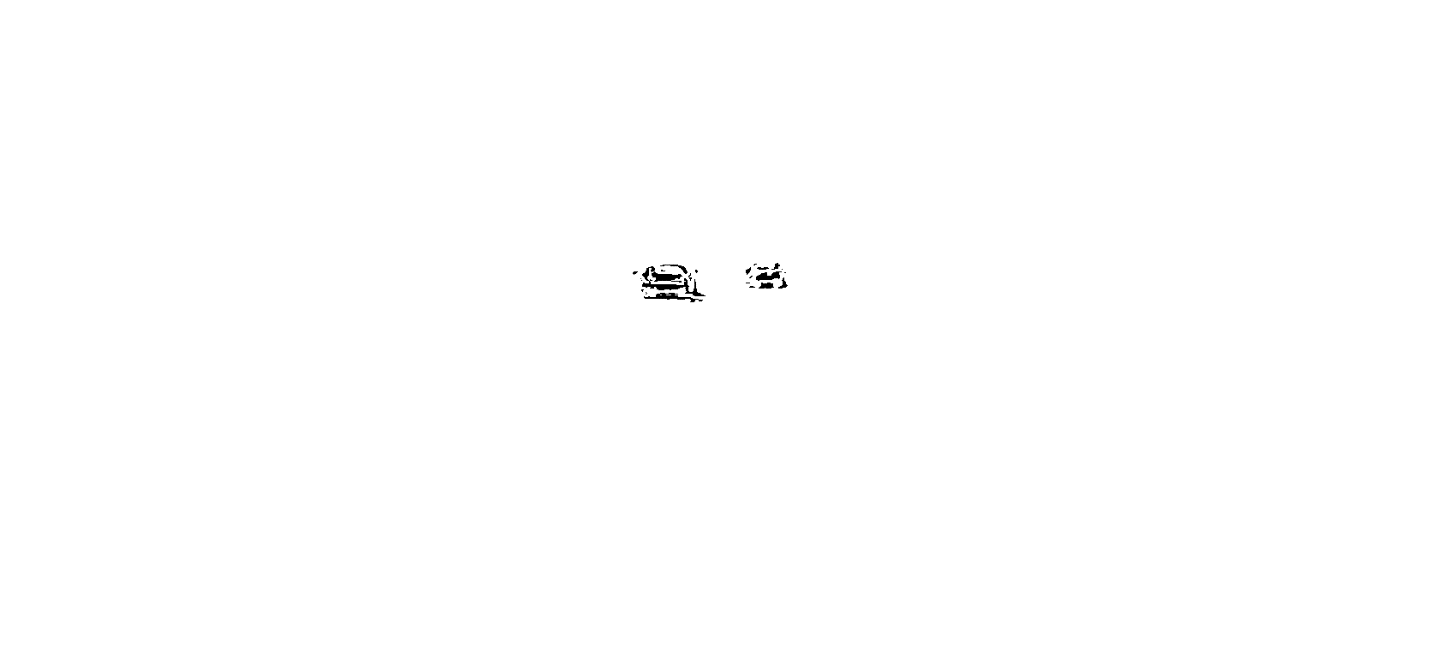


**Рисунок 2 – Получение препарата межкадровой разности и его бинаризация;**

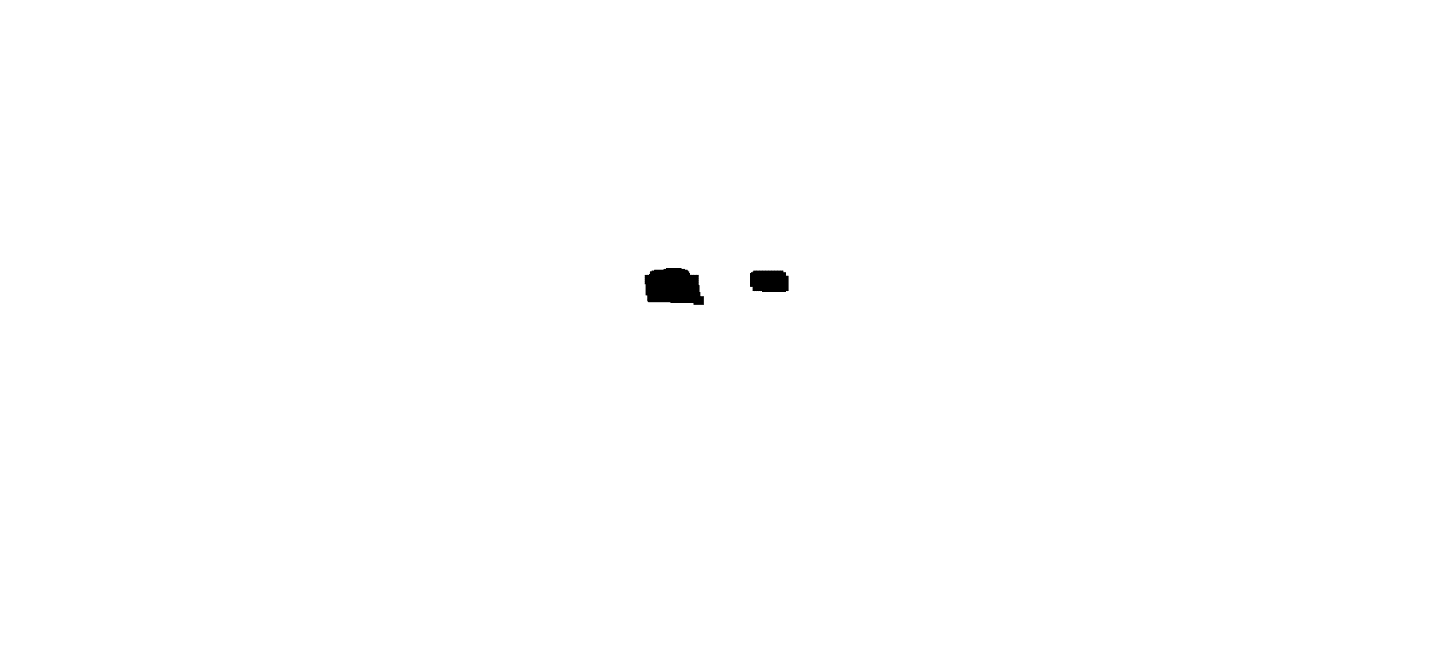




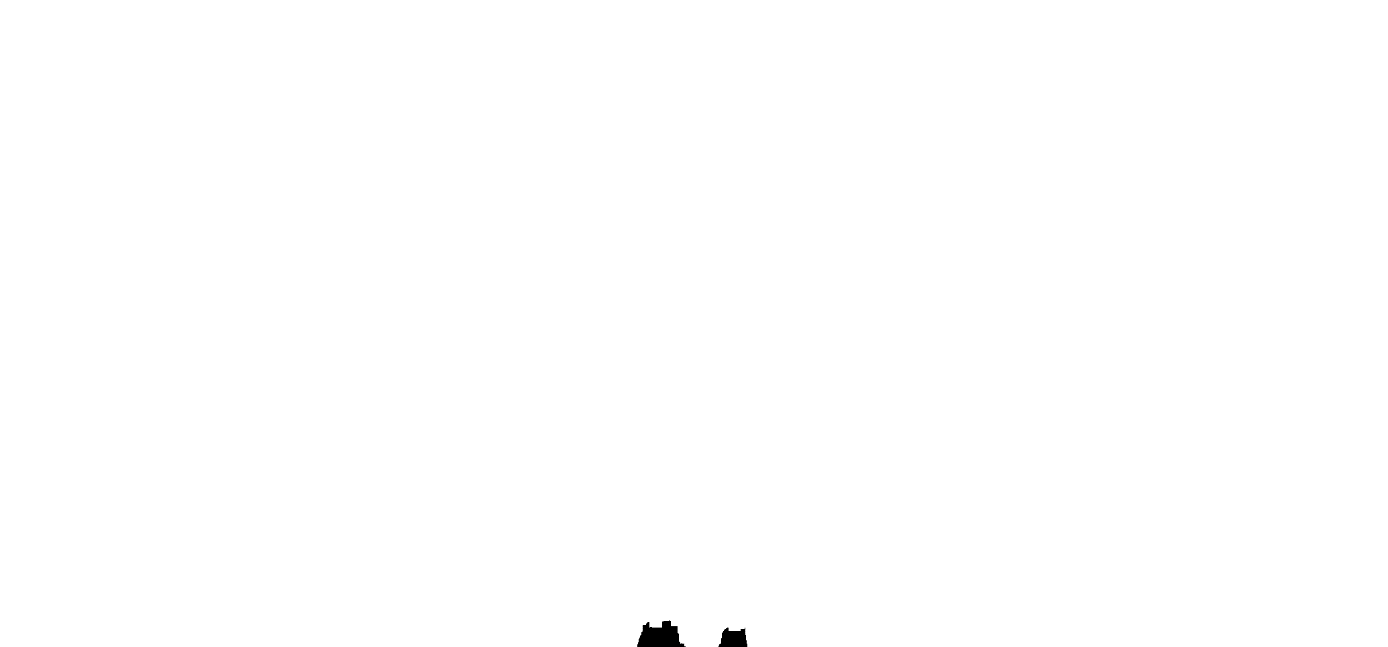
**Рисунок 3 – Выделение контуров и бинаризация**



**Рисунок 4 – Операция логического «И»**



**Рисунок 5 – Операция закрытия**

****

**Рисунок 7 – Гистограммы**

****

**Рисунок 8 – Стробирование**

**5. Листинг программы:**

|  |
| --- |
| # include < vector> |
|  | #include <iostream> |
|  | # include < opencv2/core.hpp> |
|  | # include < opencv2/imgproc.hpp> |
|  | # include < opencv2/highgui.hpp> |
|  |  |
|  | using namespace std; |
|  | using namespace cv; |
|  |  |
|  | // -- |
|  | void erosion(const Mat &input\_img, Mat &output\_img, int apert) |
|  | { |
|  | output\_img = Mat::zeros( input\_img.size(), input\_img.type() ); |
|  | for( int i = (apert / 2); i < input\_img.cols - (apert / 2); i++) { |
|  | for(int j = (apert / 2); j < input\_img.rows - (apert / 2); j++) { |
|  | // uchar pix\_value = input\_img.at<uchar>(j, i); |
|  | uchar min = 255; |
|  | for(int ii = 0 - (apert / 2); ii <= (apert / 2 ); ii++) { |
|  | for(int jj = 0 - (apert / 2 ); jj <= (apert / 2 ); jj++) { |
|  | uchar Y = input\_img.at<uchar>(j + jj, i + ii); |
|  | if (Y < min) min = Y; |
|  | } |
|  | } |
|  | output\_img.at<uchar>(j, i) = min; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // ++ |
|  | void dilatac(const Mat &input\_img, Mat &output\_img, int apert) |
|  | { |
|  | output\_img = Mat::zeros( input\_img.size(), input\_img.type() ); |
|  | for (int i = (apert / 2); i < input\_img.cols - (apert / 2); i++) { |
|  | for (int j = (apert / 2); j < input\_img.rows - (apert / 2); j++) { |
|  | //uchar pix\_value = input\_img.at<uchar>(j, i); |
|  | uchar max = 0; |
|  | for (int ii = 0 - (apert / 2); ii <= (apert / 2); ii++) { |
|  | for (int jj = 0 - (apert / 2); jj <= (apert / 2); jj++) { |
|  | uchar Y = input\_img.at<uchar>(j + jj, i + ii); |
|  | if (Y > max) max = Y; |
|  | } |
|  | } |
|  | output\_img.at<uchar>(j, i) = max; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void Roberts(const Mat &input\_img, Mat &output\_img) |
|  | { |
|  | output\_img = Mat::zeros( input\_img.size(), input\_img.type() ); |
|  | Mat Rf1 = ( Mat\_< char >(3,3) << 1, 1, 1, |
|  | 1, -2, 1, |
|  | -1, -1,-1 ); |
|  | Mat Rf2 = ( Mat\_< char >(3,3) << -1, 1, 1, |
|  | -1, -2, 1, |
|  | -1, 1, 1 ); |
|  | Mat Rf3 = ( Mat\_< char >(3,3) << -1, -1, 1, |
|  | -1, -2, 1, |
|  | 1, 1, 1 ); |
|  |  |
|  | Mat img\_Rf1, img\_Rf2, img\_Rf3; |
|  |  |
|  | filter2D( input\_img, img\_Rf1, -1, Rf1 ); |
|  | filter2D( input\_img, img\_Rf2, -1, Rf2 ); |
|  | filter2D( input\_img, img\_Rf3, -1, Rf3 ); |
|  |  |
|  | addWeighted( img\_Rf1, 1.0, img\_Rf2, 1.0, 0, output\_img ); |
|  | addWeighted( output\_img, 1.0, img\_Rf3, 1.0, 0, output\_img ); |
|  | } |
|  |  |
|  | void Sobel(const Mat &input\_img, Mat &output\_img) |
|  | { |
|  | output\_img = Mat::zeros( input\_img.size(), input\_img.type() ); |
|  | int Fk[3][3] = { { 1,1,1 }, |
|  | { 1,1,1 }, |
|  | { 1,1,1 } }; // маска фильтра 3\*3 |
|  |  |
|  | for (int i = 1; i < input\_img.cols - 1; i++) |
|  | for (int j = 1; j < input\_img.rows - 1; j++) { |
|  | // далее производим свертку |
|  | for (int ii = -1; ii <= 1; ii++) |
|  | for (int jj = -1; jj <= 1; jj++) { |
|  | //uchar blurred = input\_img.at<uchar>(j + jj, i + ii); |
|  | Fk[ii + 1][jj + 1] = input\_img.at< uchar >(j + jj, i + ii); |
|  | } |
|  | int blurred = abs((Fk[0][0] + 2 \* Fk[0][1] + Fk[0][2]) - (Fk[2][0] + 2 \* Fk[2][1] + Fk[2][2])) + |
|  | abs((Fk[0][0] + 2 \* Fk[1][0] + Fk[2][0]) - (Fk[0][2] + 2 \* Fk[1][2] + Fk[2][2])); |
|  | if (blurred < 0) blurred = 0; |
|  | else if (blurred > 255) blurred = 255; |
|  | //else blurred = blurred; |
|  | output\_img.at< uchar >(j, i) = uchar(blurred); |
|  | } |
|  | } |
|  |  |
|  | void Hist( Mat img, Mat &hh, Mat &hv) |
|  | { |
|  | hh = Mat::ones( img.size(), img.type() ); |
|  | hv = Mat::ones( img.size(), img.type() ); |
|  | hh \*= 255; |
|  | hv \*= 255; |
|  |  |
|  | bitwise\_not( img, img ); |
|  |  |
|  | int intensity = 0; |
|  | for ( int i = 0; i < img.cols; i++ ) |
|  | { |
|  | intensity = countNonZero( img.col(i) ); |
|  | for ( int k = intensity; k != 0; k-- ) |
|  | hh.at< uchar >(hh.rows - k, i) = 0; |
|  | } |
|  | for ( int j = 0; j < img.rows; j++ ) |
|  | { |
|  | intensity = countNonZero( img.row(j) ); |
|  | for ( int k = intensity; k != 0; k-- ) |
|  | hv.at< uchar >(j, hv.cols - k) = 0; |
|  | } |
|  | } |
|  |  |
|  | Mat draw\_arctangle\_new( const Mat img\_horizon, const Mat img\_vertical, Mat img ) |
|  | { |
|  | int level1 = 1; |
|  | int level2 = 1; |
|  | vector< int > X; |
|  | vector< int > Y; |
|  |  |
|  | bitwise\_not( img\_horizon, img\_horizon ); |
|  | bitwise\_not( img\_vertical, img\_vertical ); |
|  | bool truelevel = false; |
|  | for ( int i = 0; i < img.cols; i++ ) |
|  | { |
|  | int intensity = countNonZero( img\_horizon.col(i) ); |
|  | if ( (intensity >= level1) && (!truelevel) ) |
|  | { |
|  | X.push\_back( i ); |
|  | truelevel = true; |
|  | } |
|  | else if ( (intensity < level1) && (truelevel) ) |
|  | { |
|  | X.push\_back( i-1 ); |
|  | truelevel = false; |
|  | } |
|  | if ( (intensity >= level1) && (i == img.cols-1) ) |
|  | X.push\_back( i ); |
|  | } |
|  | truelevel = false; |
|  | for ( int j = 0; j < img.rows; j++ ) |
|  | { |
|  | int intensity = countNonZero( img\_vertical.row(j) ); |
|  | if ( (intensity >= level2) && (!truelevel) ) |
|  | { |
|  | Y.push\_back( j ); |
|  | truelevel = true; |
|  | } |
|  | else if ( (intensity < level2) && (truelevel) ) |
|  | { |
|  | Y.push\_back( j-1 ); |
|  | truelevel = false; |
|  | } |
|  | if ( (intensity >= level2) && (j == img.rows-1) ) |
|  | Y.push\_back( j ); |
|  | } |
|  | rectangle(img, Point(X.at(0), Y.at(0)), Point(X.at(1), Y.at(1)), Scalar(0, 0, 255), 2, 8, 0); |
|  | rectangle(img, Point(X.at(2), Y.at(0)), Point(X.at(3), Y.at(1)), Scalar(0, 0, 255), 2, 8, 0); |
|  | return img; |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | QCoreApplication a(argc, argv); |
|  |  |
|  | // Read image |
|  | Mat img\_in1 = imread( "image\_1.png", 3 ); |
|  | Mat img\_in2 = imread( "image\_2.png", 3 ); |
|  |  |
|  | // Crop |
|  | img\_in1 = img\_in1( Rect( 200, 100, img\_in1.cols - 400, img\_in1.rows - 200 ) ); |
|  | img\_in2 = img\_in2( Rect( 200, 100, img\_in2.cols - 400, img\_in2.rows - 200 ) ); |
|  |  |
|  | // Convert to gray |
|  | Mat img\_grey1, img\_grey2; |
|  | cvtColor( img\_in1, img\_grey1, COLOR\_BGR2GRAY ); |
|  | cvtColor( img\_in2, img\_grey2, COLOR\_BGR2GRAY ); |
|  | imwrite( "img\_grey1.png", img\_grey1 ); |
|  | imwrite( "img\_grey2.png", img\_grey2 ); |
|  |  |
|  | // Gauss |
|  | int sigma = 3; |
|  | int ksize = 5; //( sigma\*5 ) | 1; |
|  | GaussianBlur( img\_grey1, img\_grey1, Size( ksize, ksize ), sigma, sigma, cv::BORDER\_DEFAULT ); |
|  | GaussianBlur( img\_grey2, img\_grey2, Size( ksize, ksize ), sigma, sigma, cv::BORDER\_DEFAULT ); |
|  | imwrite( "img\_gauss1.png", img\_grey1 ); |
|  | imwrite( "img\_gauss2.png", img\_grey2 ); |
|  |  |
|  | // Calculate difference between image |
|  | Mat img\_diff; |
|  | // img\_diff = img\_grey1.clone(); |
|  | // for ( int i = 0; i < img\_diff.total(); i++ ) |
|  | // img\_diff.at< uchar >(i) = abs(img\_grey2.at< uchar >(i) - img\_grey1.at< uchar >(i)); |
|  | absdiff( img\_grey1, img\_grey2, img\_diff ); |
|  | imwrite( "img\_diff.png", img\_diff ); |
|  |  |
|  | // Edge |
|  | Mat img\_edge1; |
|  | // Canny( img\_grey1, img\_edge1, 70, 150, 3, false ); |
|  | // Sobel( img\_grey1, img\_edge1 ); |
|  | Roberts( img\_grey1, img\_edge1 ); |
|  | imwrite( "img\_edge1.png", img\_edge1 ); |
|  |  |
|  | // Binarization |
|  | Mat img\_edge1\_bin, img\_diff\_bin; |
|  | vector< int > params\_write; |
|  | params\_write.push\_back( IMWRITE\_PNG\_BILEVEL ); |
|  |  |
|  | threshold( img\_diff, img\_diff\_bin, 10, 255, THRESH\_BINARY ); |
|  | Mat idbe; |
|  | erosion( img\_diff\_bin, idbe, 3 ); |
|  | idbe.copyTo( img\_diff\_bin ); |
|  | imwrite( "img\_diff\_bin.png", img\_diff\_bin ); |
|  |  |
|  | threshold( img\_edge1, img\_edge1\_bin, 10, 255, THRESH\_BINARY ); // THRESH\_BINARY\_INV THRESH\_BINARY |
|  | imwrite( "img\_edge1\_bin.png", img\_edge1\_bin ); |
|  |  |
|  | // Logical AND |
|  | Mat img\_and = img\_diff.clone(); |
|  | // for ( int i = 0; i < int(img\_and.total()); i++ ) |
|  | // img\_and.at< uchar >(i) = img\_diff\_bin.at< uchar >(i) & img\_edge1\_bin.at< uchar >(i); |
|  | bitwise\_and( img\_diff\_bin, img\_edge1\_bin, img\_and ); |
|  | bitwise\_not( img\_and, img\_and ); // invertion image |
|  | imwrite( "img\_and.png", img\_and ); // , params\_write |
|  |  |
|  | // Morphological operations |
|  | Mat img\_temp = Mat::zeros( img\_and.size(), img\_and.type() ); |
|  | Mat img\_morf = Mat::zeros( img\_and.size(), img\_and.type() ); |
|  | Mat element = getStructuringElement( MORPH\_RECT, Size(10, 10), Point(-1, -1) ); |
|  | morphologyEx( img\_and, img\_temp, MORPH\_OPEN, element ); |
|  | morphologyEx( img\_temp, img\_morf, MORPH\_CLOSE, element ); |
|  | imwrite( "img\_morfolog\_1.png", img\_morf ); |
|  | // morphologyEx( img\_morf, img\_temp, MORPH\_OPEN, element ); |
|  | // morphologyEx( img\_temp, img\_morf, MORPH\_CLOSE, element ); |
|  | // imwrite( "img\_morfolog\_2.png", img\_morf ); |
|  |  |
|  | // Histogram |
|  | Mat hist\_vertical, hist\_horizon; |
|  | Hist( img\_morf, hist\_horizon, hist\_vertical ); |
|  | imwrite( "hist\_horizon.png", hist\_horizon ); |
|  | imwrite( "hist\_vertical.png", hist\_vertical ); |
|  |  |
|  | // Strobe |
|  | Mat img\_finish; |
|  | img\_finish = draw\_arctangle\_new( hist\_horizon, hist\_vertical, img\_in1 ); |
|  | imwrite( "finish.png", img\_finish ); |
|  |  |
|  | waitKey(0); |
|  | return 0; |
|  | } |