**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

отчет

**по лабораторной работе №2**

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

Тема: **СЕГМЕНТАЦИЯ ДВИЖУЩИХСЯ ОБЪЕКТОВ НА ОСНОВЕ**

**ОЦЕНКИ ПОЛЯ ВЕКТОРОВ ДВИЖЕНИЯ**

|  |  |  |
| --- | --- | --- |
| 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 на основе оценки поля векторов 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,

расчет векторов движения (логарифмическим методом, MSE оценка),

пространственная фильтрация поля векторов движения,

кластеризация блоков, для которых определены ненулевые векторы движения, определение их принадлежности к соответствующим объектам.

**3. Краткие теоретические сведения:**

Для определения векторов движения основным является уравнение оптического потока, полученное на основе допущения о постоянстве яркости L(x,y,t) точки (пикселя) при движении.

Для поиска векторов движения существует три группы методов: дифференциальные, фазовые и корреляционные.

Формирование целевой функции, оценивающей степень соответствия между блоком текущего кадра и блоком предыдущего кадра, может быть выполнено в нескольких вариантах: MAD, MSE, NCCF и MPC.

Совмещение блоков. Самым простым и надежным алгоритмом, позволяющим выполнить совмещение блоков, является полный перебор FS (full search), но из-за большого объема вычислений он обладает низкой скоростью.

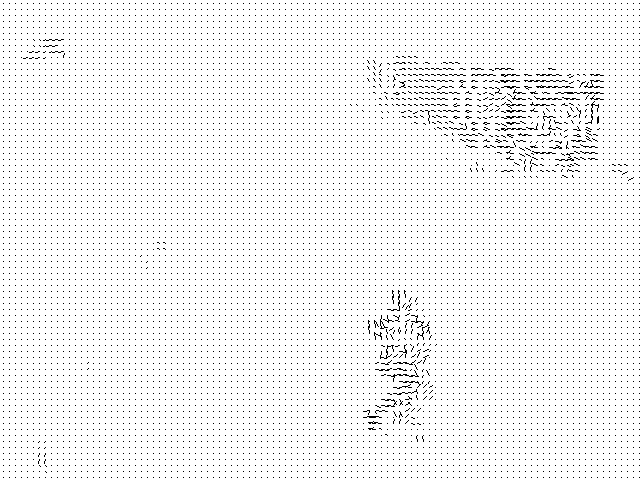
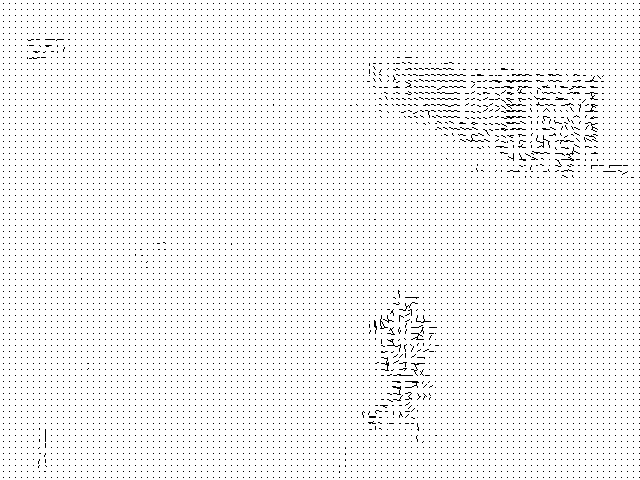
На заключительном этапе алгоритма сегментации объектов на основе векторов движения найденные векторы, прошедшие фильтрацию, объединяют в кластеры по признаку сонаправленности и пространственной связанности.

**4. Результаты работы:**

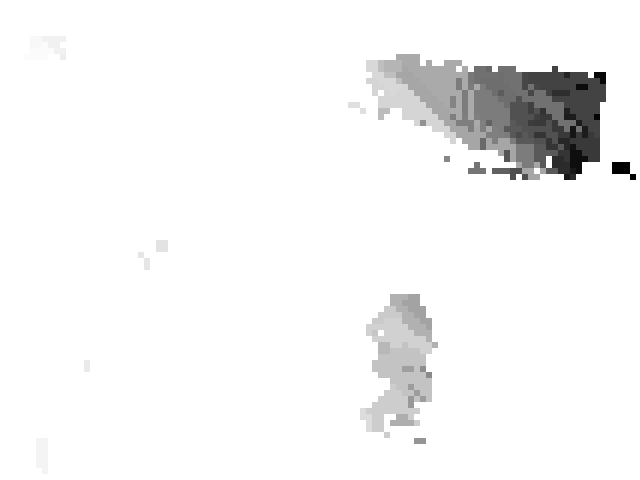
Входные изображения:

****

Производим поиск векторов движения, алгоритм смещения блоков — логарифмический, целевая функция — MSE, размер блока — 6х6:

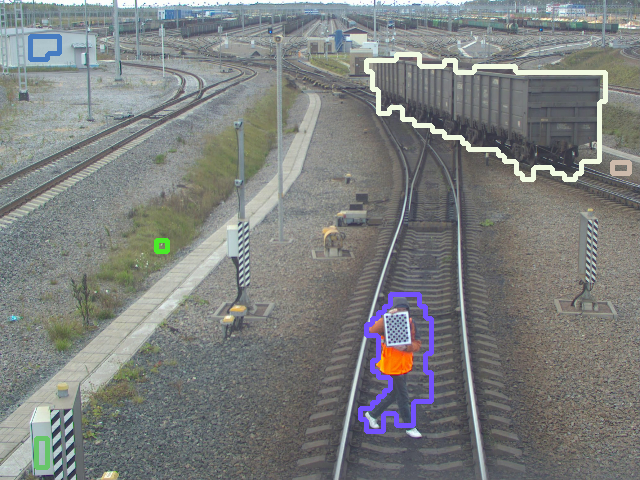
Рекурсивная векторная медианная фильтрация:

Кластеризуем:

Немного морфологии:



Выделяем контура объектов:



**5. Листинг программы:**

#include <vector>

#include <queue>

#include <math.h>

#include <iostream>

#include <opencv2/core.hpp>

#include <opencv2/imgproc.hpp>

#include <opencv2/highgui.hpp>

#include <opencv2/core/hal/interface.h>

#include <opencv2/optflow.hpp>

*using* *namespace* std;

*using* *namespace* cv;

#define RESIZE 1

#define W\_img 640 *//* *640* *320*

#define H\_img 480 *//* *480* *240*

*//* *Структура* *для* *фильтрации* *векторов*

*struct* **L2\_norm**

{

float L2;

Point dxy;

Point xy;

};

*//* *Перегруженный* *оператор* *для* *сортировки*

*struct* **less\_then\_L2**

{

*inline* bool *operator*() (*const* L2\_norm& str1, *const* L2\_norm& str2)

{

*return* (str1.L2 < str2.L2);

}

};

void **ReadVideo**( string fn, Mat \*img1, Mat \*img2 )

{

VideoCapture cap( fn );

*if*( !cap.*isOpened*() )

*throw* "Error when reading " + fn;

cap.*set*( *CAP\_PROP\_POS\_MSEC*, 309000 ); *//* *3300* *136300* *309000*

Mat frameRAW;

*while*(1)

{

*if* ( !cap.*read*( frameRAW ) ) {

cerr << "ERROR! blank frame grabbed\n";

*break*;

}

#if (RESIZE == 1)

resize( frameRAW, frameRAW, Size(W\_img, H\_img), 0, 0, *INTER\_LINEAR*);

#endif

imshow( "Real time", frameRAW );

int button = waitKey();

*if* ( (button == 13) || (button == 141) ) *//* *Take* *picture,* *press* *"Enter"*

{

frameRAW.copyTo( \*img1 );

cap.*read*( frameRAW );

#if (RESIZE == 1)

resize( frameRAW, frameRAW, Size(W\_img, H\_img), 0, 0, *INTER\_LINEAR*);

#endif

frameRAW.copyTo( \*img2 );

}

*else* *if* ( button == 27 )

{

cap.*release*();

destroyWindow( "Real time" );

*break*; *//* *Interrupt* *the* *cycle,* *press* *"ESC"*

}

}

}

void **PaintVectors**( Mat &flow, Mat &vectors, *const* Size MN )

{

*for* ( int x = 0; x < flow.cols; x += MN.width )

{

*for* ( int y = 0; y < flow.rows; y += MN.height )

{

*const* Point2f flowatxy = flow.at< Point2f >(y, x) \* 1;

line( vectors,

Point( x + MN.width/2, y + MN.height/2 ),

Point( cvRound( x + flowatxy.x + MN.width/2 ), cvRound(y + flowatxy.y + MN.height/2 ) ),

Scalar(255, 255, 255) );

}

}

}

void **LogMSE**( *const* Size MN, vector< Point2i > V, Point2i bp, Point2i delta, int vS, Mat img1, Mat img2, Mat &mse )

{

float square = MN.width \* MN.height; *//* *square* *block*

mse = Mat::ones( mse.size(), CV\_32FC1 );

mse \*= 65025;

vector< float > mse\_i( V.size(), 0.0f );

*for* ( size\_t k = 0; k < V.size(); k++ ) *//* *num* *block*

{

*if* ( ( delta.x + vS\*V[k].x <= mse.cols ) && ( delta.y + vS\*V[k].y <= mse.rows ) &&

( delta.x + vS\*V[k].x >= 0 ) && ( delta.y + vS\*V[k].y >= 0 ) ) *//* *border* *check*

{

*for* ( int x = 0; x < MN.width; x++ ) *//* *cols*

{

*for* ( int y = 0; y < MN.height; y++ ) *//* *rows*

{

mse\_i[k] += (img1.at< uchar >(bp.y + y, bp.x + x) - img2.at< uchar >(delta.y + vS\*V[k].y + y, delta.x + vS\*V[k].x + x)) \*

(img1.at< uchar >(bp.y + y, bp.x + x) - img2.at< uchar >(delta.y + vS\*V[k].y + y, delta.x + vS\*V[k].x + x));

*//* *mse\_i[k]* *+=* *abs(img1.at<* *uchar* *>(bp.y* *+* *y,* *bp.x* *+* *x)* *-* *img2.at<* *uchar* *>(delta.y* *+* *vS\*V[k].y* *+* *y,* *delta.x* *+* *vS\*V[k].x* *+* *x));*

}

}

mse.at< float >(delta.y + vS\*V[k].y, delta.x + vS\*V[k].x) = mse\_i[k] / square;

}

}

}

void **filter\_vectors**( Mat &src, Mat &dst, *const* Size MN )

{

*const* vector< Point2i > V = { Point(-1,-1), Point(0,-1), Point(1,-1), Point(-1,0), Point(0,0),

Point(1,0), Point(-1,1), Point(0,1), Point(1,1) };

*//* *Recursive* *vector* *median* *filtering*

*//dst* *=* *src.clone();*

*for* ( int x = 0; x < src.cols; x += MN.width )

{

*for* ( int y = 0; y < src.rows; y += MN.height )

{

*//* *проверка* *нулевости* *вектора*

*if* ( !(src.at< Point2f >(y, x) == Point2f(0, 0)) )

{

*//* *блок* *из* *суммы* *вычесленных* *растояний* *между* *векторами*

vector< L2\_norm > d\_block;

*for* ( size\_t k = 0; k < 9; k++ ) *//* *num* *block,* *блоки* *смещения*

{

L2\_norm di;

di.L2 = 0;

di.xy = Point(0,0);

di.dxy = Point(0,0);

*//* *Border* *check*

*if* ( ( x + MN.width\*V[k].x <= src.cols ) && ( y + MN.height\*V[k].y <= src.rows ) &&

( x + MN.width\*V[k].x >= 0 ) && ( y + MN.height\*V[k].y >= 0 ) )

{

*//* *save* *value* *vectors*

di.xy = Point( x + MN.width\*V[k].x, y + MN.height\*V[k].y ); *//* *point*

di.dxy = Point( int( src.at< Point2f >(y + MN.height\*V[k].y, x + MN.width\*V[k].x).x ),

int( src.at< Point2f >(y + MN.height\*V[k].y, x + MN.width\*V[k].x).y ) ); *//* *offset*

*for* ( size\_t kk = 0; kk < 9; kk++ ) *//* *2й* *блок* *сещения*

{

*//* *пропускаем* *схожие* *смещения*

*if* ( V[k] != V[kk] )

{

*//* *Border* *check*

*if* ( ( x + MN.width\*V[kk].x <= src.cols ) && ( y + MN.height\*V[kk].y <= src.rows ) &&

( x + MN.width\*V[kk].x >= 0 ) && ( y + MN.height\*V[kk].y >= 0 ) )

{

*//* *Проверка* *нулевости* *смещенного* *вектора*

*if* ( !(src.at< Point2f >(y + MN.height\*V[k].y, x + MN.width\*V[k].x) == Point2f(0, 0)) &&

!(src.at< Point2f >(y + MN.height\*V[kk].y, x + MN.width\*V[kk].x) == Point2f(0, 0)) )

{

di.L2 += sqrt( (((src.at< Point2f >(y + MN.height\*V[k].y, x + MN.width\*V[k].x).x - (x + MN.width\*V[k].x)) - (src.at< Point2f >(y + MN.height\*V[kk].y, x + MN.width\*V[kk].x).x - (x + MN.width\*V[kk].x))) \*

((src.at< Point2f >(y + MN.height\*V[k].y, x + MN.width\*V[k].x).x - (x + MN.width\*V[k].x)) - (src.at< Point2f >(y + MN.height\*V[kk].y, x + MN.width\*V[kk].x).x - (x + MN.width\*V[kk].x)))) +

(((src.at< Point2f >(y + MN.height\*V[k].y, x + MN.width\*V[k].x).y - (y + MN.height\*V[k].y)) - (src.at< Point2f >(y + MN.height\*V[kk].y, x + MN.width\*V[kk].x).y - (y + MN.height\*V[kk].y))) \*

((src.at< Point2f >(y + MN.height\*V[k].y, x + MN.width\*V[k].x).y - (y + MN.height\*V[k].y)) - (src.at< Point2f >(y + MN.height\*V[kk].y, x + MN.width\*V[kk].x).y - (y + MN.height\*V[kk].y)))) );

}

}

}

}

}

d\_block.push\_back(di);

}

*//* *Сортировка* *по* *норме* *L2*

sort( d\_block.begin(), d\_block.end(), less\_then\_L2() );

*for* ( size\_t i = 0; i < 9; i++ )

{

*//* *Отбрасываем* *нулевые* *элементы*

*if* ( d\_block.at(i).dxy.x != 0 && d\_block.at(i).dxy.y != 0) *//(* *Point(d\_block.at(i).dxy.x,* *d\_block.at(i).dxy.y)* *!=* *Point(0,* *0)* *)* *//(* *(d\_block.at(i).dxy.x* *+* *d\_block.at(i).dxy.y)* *>* *0.0* *)*

{

*//* *Запись* *только* *в* *центрльную* *ячейку*

float xdx = d\_block.at(i).dxy.x;

float ydy = d\_block.at(i).dxy.y;

*if* ( (x + xdx >= 0) && (x + xdx <= dst.cols) && (y + ydy >= 0) && (y + ydy <= dst.rows) )

dst.at< Point2f >(y, x) = Point2f( xdx, ydy );

*break*;

}

}

}

}

}

}

int **main**() *//* *int* *argc,* *char* *\*argv[]*

{

*//* *Read* *image*

*//string* *file\_name* *=* *"video.avi";*

string file\_name = "/home/roman/Reconst\_Stereo/zcm\_logs/zcmlog-2019-09-23\_L\_05.avi";

Mat img\_in1, img\_in2;

ReadVideo( file\_name, &img\_in1, &img\_in2 );

*if* ( img\_in1.empty() || img\_in2.empty() )

{

cout << " ! ! ! - Image is empty" << endl;

exit(0);

}

imwrite( "img\_in1.png", img\_in1 );

imwrite( "img\_in2.png", img\_in2 );

*//* *Convert* *to* *gray* *and* *blur*

Mat img\_grey1, img\_grey2;

cvtColor( img\_in1, img\_grey1, *COLOR\_BGR2GRAY* );

cvtColor( img\_in2, img\_grey2, *COLOR\_BGR2GRAY* );

GaussianBlur( img\_grey1, img\_grey1, Size(3,3), 7, 0, *BORDER\_DEFAULT*);

GaussianBlur( img\_grey2, img\_grey2, Size(3,3), 7, 0, *BORDER\_DEFAULT*);

imwrite( "img\_grey1.png", img\_grey1 );

imwrite( "img\_grey2.png", img\_grey2 );

cout << "img.cols (width) = " << img\_in1.cols << endl;

cout << "img.rows (height) = " << img\_in1.rows << endl;

*//* *Input* *block* *size* *(M* *x* *N)*

Size S\_block;

cout << "input M = ";

cin >> S\_block.width;

cout << "input N = ";

cin >> S\_block.height;

cout << "Block(M x N)= " << S\_block.width << "x" << S\_block.height << endl;

*if* ( S\_block.width == 0 || S\_block.height == 0 )

{

cout << " ! ! ! - M & N mast be > 0" << endl;

exit(0);

}

*//* *MSE* *&* *optical* *vectors*

Mat flow = Mat::zeros( img\_grey1.size(), CV\_32FC2 );

Mat MSE = Mat::zeros( img\_grey1.size(), CV\_32FC1 );

int start\_num = 4;

int Size\_vector = start\_num; *//* *Initial* *displacement* *vector*

double min\_MSE = 0, max\_MSE = 65025;

Point2i min\_MSE\_point, max\_MSE\_point;

Point2i offset\_point, block\_point;

*//* *Possible* *displacement* *vectors* *in* *a* *logarithmic* *search*

*const* vector< Point2i > Vcross = { Point(0,0), Point(1,0), Point(0,1), Point(-1,0), Point(0,-1) };

*const* vector< Point2i > Vsquare = { Point(0,0), Point(1,0), Point(1,1), Point(0,1), Point(-1,1),

Point(-1,0), Point(-1,-1), Point(0,-1), Point(1,-1) };

*//* *Calculate* *vectors*

float progress = 0.0;

int barWidth = 70;

float step = 1.0f / flow.total() \* S\_block.width \* S\_block.height;

progress += step;

cout << " --- Start vector calculation" << endl;

*for* ( int i = 0; i < flow.cols; i += S\_block.width )

{

*for* ( int j = 0; j < flow.rows; j += S\_block.height )

{

block\_point.x = i;

block\_point.y = j;

offset\_point.x = i;

offset\_point.y = j;

*while* ( Size\_vector > 1 )

{

LogMSE( S\_block, Vcross, block\_point, offset\_point, Size\_vector, img\_grey2, img\_grey1, *MSE* );

minMaxLoc( MSE, &min\_MSE, &max\_MSE, &min\_MSE\_point, &max\_MSE\_point );

Size\_vector /= 2;

*if* ( offset\_point != min\_MSE\_point ) offset\_point = min\_MSE\_point;

}

LogMSE( S\_block, Vsquare, block\_point, offset\_point, Size\_vector, img\_grey2, img\_grey1, *MSE* );

minMaxLoc( MSE, &min\_MSE, &max\_MSE, &min\_MSE\_point, &max\_MSE\_point );

flow.at< Point2f >(j, i) = Point2f( i - min\_MSE\_point.x, j - min\_MSE\_point.y );

Size\_vector = start\_num;

*//* *progress*

progress += step;

cout << "[";

int pos = int( float(barWidth) \* progress );

*for* (int i = 0; i < barWidth; ++i) {

*if* (i < pos) cout << "=";

*else* *if* (i == pos) cout << ">";

*else* cout << " ";

}

cout << "] " << int(progress \* 100.0f) << " %\r";

cout.flush();

}

}

cout << endl;

*//optflow::calcOpticalFlowSparseToDense(* *img\_grey2,* *img\_grey1,* *flow,* *4,* *128,* *0.1f,* *true,* *500.0f,* *1.5f);*

*//* *Draw* *vectors*

Mat vectors = Mat::zeros(img\_in1.size(), img\_in1.type());

PaintVectors( *flow*, *vectors*, S\_block );

bitwise\_not( vectors, vectors );

imwrite( "Flow.png", vectors );

*//imshow(* *"vec\_flow",* *vectors* *);*

cout << " --- vec\_dif cmplited" << endl;

*//* *Recursive* *vector* *median* *filtering*

Mat flow\_new = flow.clone();

flow\_new \*= 0;

*for* ( int i = 0; i < 2; i++ )

{

filter\_vectors( *flow*, *flow\_new*, S\_block );

flow\_new.copyTo( flow );

}

*//* *Draw* *new* *vectors*

vectors = Mat::zeros(img\_in1.size(), img\_in1.type());

PaintVectors( *flow\_new*, *vectors*, S\_block );

bitwise\_not( vectors, vectors );

imwrite( "Flow\_new.png", vectors );

imshow( "vec\_flow", vectors );

cout << " --- New\_vec\_dif cmplited" << endl;

*//* *Clustering*

Mat cluster = Mat::zeros( flow\_new.size(), CV\_8UC1 );

*const* vector< Point2i > V = { Point(-1,-1), Point(0,-1), Point(1,-1), Point(-1,0), Point(0,0),

Point(1,0), Point(-1,1), Point(0,1), Point(1,1) };

uchar marker = 0;

*for* ( int x = 0; x < cluster.cols; x += S\_block.width )

{

*for* ( int y = 0; y < cluster.rows; y += S\_block.height )

{

*if* ( cluster.at< uchar >(y, x) == 0 && flow\_new.at< Point2f >(y, x) != Point2f(0, 0) )

{

marker++;

cluster.at< uchar >(y, x) = marker;

}

*for* ( size\_t k = 0; k < 9; k++ ) *//* *num* *block,* *блоки* *смещения*

{

*//* *Border* *check*

*if* ( ( x + S\_block.width\*V[k].x <= cluster.cols ) && ( y + S\_block.height\*V[k].y <= cluster.rows ) &&

( x + S\_block.width\*V[k].x >= 0 ) && ( y + S\_block.height\*V[k].y >= 0 ) )

{

*//* *not* *zero* *vector* *&* *zero* *cluster*

*if* ( flow\_new.at< Point2f >(y + S\_block.height\*V[k].y, x + S\_block.width\*V[k].x) != Point2f(0, 0) &&

cluster.at< uchar >(y + S\_block.height\*V[k].y, x + S\_block.width\*V[k].x) == 0 )

{

double V1V2 = double( ( flow\_new.at< Point2f >(y, x).x ) \*

( flow\_new.at< Point2f >(y + S\_block.height\*V[k].y, x + S\_block.width\*V[k].x).x ) +

( flow\_new.at< Point2f >(y, x).y ) \*

( flow\_new.at< Point2f >(y + S\_block.height\*V[k].y, x + S\_block.width\*V[k].x).y ) );

double VV = sqrt( (pow( flow\_new.at< Point2f >(y, x).x, 2 ) +

pow( flow\_new.at< Point2f >(y, x).y, 2 )) \*

(pow( flow\_new.at< Point2f >(y + S\_block.height\*V[k].y, x + S\_block.width\*V[k].x).x, 2 ) +

pow( flow\_new.at< Point2f >(y + S\_block.height\*V[k].y, x + S\_block.width\*V[k].x).y, 2 )) );

double sinA = V1V2 / VV; *//* *Range* *-1..1*

*//* *Angle* *check*

*if* ( sinA > 0 )

cluster.at< uchar >(y + S\_block.height\*V[k].y, x + S\_block.width\*V[k].x) = cluster.at< uchar >(y, x);

}

}

}

}

}

cout << "Number of cluster: " << marker \* 1.0 << endl;

normalize( cluster, cluster, 255.0, 0.0, *NORM\_MINMAX* );

*for* ( int x = 0; x < cluster.cols; x += S\_block.width )

*for* ( int y = 0; y < cluster.rows; y += S\_block.height )

*if* ( cluster.at< uchar >(y, x) )

*for* ( int i = 0; i < S\_block.width; i++ )

*for* ( int j = 0; j < S\_block.height; j++ )

cluster.at< uchar >(y+j, x+i) = cluster.at< uchar >(y, x);

Mat img\_temp = cluster.clone();

bitwise\_not( cluster, img\_temp );

imshow( "Cluster", img\_temp );

imwrite( "Cluster.png", img\_temp );

threshold( cluster, cluster, 1, 255, *THRESH\_BINARY* );

Mat element = getStructuringElement( *MORPH\_RECT*, Point(int(S\_block.width\*1.3), int(S\_block.height\*1.3)), Point(-1, -1) );

morphologyEx( cluster, img\_temp, *MORPH\_OPEN*, element );

morphologyEx( img\_temp, cluster, *MORPH\_CLOSE*, element );

bitwise\_not( cluster, img\_temp );

imshow( "Morph", img\_temp );

imwrite( "Morph.png", img\_temp );

vector< vector< Point > > contours;

vector< Vec4i > hierarchy;

findContours( cluster, contours, hierarchy, *RETR\_LIST*, *CHAIN\_APPROX\_SIMPLE* );

int idx = 0;

*for*( ; idx >= 0; idx = hierarchy[unsigned(idx)][0] )

{

Scalar color( rand()&255, rand()&255, rand()&255 );

drawContours( img\_in1, contours, idx, color, 3, *LINE\_8*, hierarchy );

}

imshow( "Conturs", img\_in1 );

imwrite( "Conturs.png", img\_in1 );

waitKey(0);

MSE.release();

flow.release();

*return* 0;

}