Contents

[**Этап 1. Уточнение задачи.** 2](#_Toc277166037)

[**Этап 2. Последовательность выполнения задач** 3](#_Toc277166038)

[Начало потока(1) Обработка видеоданных. 3](file:///C:\Users\Lenin133\Desktop\Задача%20№1.docx#_Toc277166039)

[Начало потока(2) Определение движения наблюдателя 4](file:///C:\Users\Lenin133\Desktop\Задача%20№1.docx#_Toc277166040)

[Начало потока(3) Распознавание объектов. 4](file:///C:\Users\Lenin133\Desktop\Задача%20№1.docx#_Toc277166041)

[Начало потока(4) Привязка к ГСК 5](file:///C:\Users\Lenin133\Desktop\Задача%20№1.docx#_Toc277166042)

[**Этап 3. Анализ данных.** 6](#_Toc277166043)

[Данные потока(1): 6](#_Toc277166044)

[***Время.*** 6](#_Toc277166045)

[***Видеоданные: частота кадров, потоки RGB данных.*** 6](#_Toc277166046)

[***Эквализация*** 8](#_Toc277166047)

[***Дискретное косинусное преобразование.*** 9](#_Toc277166048)

[***Преобразование Фурье*** 11](#_Toc277166049)

[***Матрицы фильтра Гаусса разного масштаба*** 15](#_Toc277166050)

[***Свертка*** 16](#_Toc277166051)

[***Нормализованный Лапласиан по масштабированным Гауссианам*** 17](#_Toc277166052)

[***Набор характеристтик области внимания*** 20](#_Toc277166053)

[Данные потока(2): 23](#_Toc277166054)

[***Аффинное преобразование*** 23](#_Toc277166055)

[***Определение матрицы аффинного преобразования*** 24](#_Toc277166056)

[***Определение общего нарастающего изменения*** 26](#_Toc277166057)

[Данные потока(3): 27](#_Toc277166058)

[***Корреляция*** 27](#_Toc277166059)

**Этап 1. Уточнение задачи.**

1. Определение изменения положения наблюдающего устройства в пространстве.
2. Повышение качества изображения
3. Распознавание объектов на изображении
4. Привязка ортофотоплана к глобальной системе координат

**Этап 2. Последовательность выполнения задач**

Начало потока(1) Обработка видеоданных.

Выделение характерных особенностей, в области внимания, выбранных объектов.

Источник видеоданных: MPEG, AVI, BMP, JPEG

1

1

Карта характеристик объектов, через поток распознавания

Карта объектов, через поток привязки к ГСН

Определение границ объектов. Составление карты объектов. Выбор области внимания.

БД, карт характерных особенностей объектов, раскадровки, глобальной привязки объектов

Конец потока(1) подготовки данных

Дешифровка видеопотока.

Фильтрация

1

Стандартное отклонение, ассимметрия, межпиксельная контрасность, бета, горизонтальный градиент, вертикальный градиент, максимальны градиент, взаимное расположение

Сегментация, водораздел, поиск углов, скрещивание фильтров

(R, G, B, Y)

L, Lx, Ly, Lxy, Lxx, Lyy, Lxxyy

Lnorm, Sx, Sy, Sxy, H

Re, Ge, Be, Ye

FFT, DCT

Конец потока(2)

Начало потока(2) Определение движения наблюдателя

Определение матрицы движения в пространстве. Фильтрация шума по инерционной модели.

1

Найденные объекты. Координаты центров, в относительных и глобальных СК

Конец потока(3)

Фильтр шума. Статистика.

Запрос информации по новым объектам у оператора.

Выбор максимально похожего, оценка степени сходимости.

Сравнение параметров области внимания с картами характерных особенностей извесных объектов.

Начало потока(3) Распознавание объектов.

Координаты центров распознанных объектов, параметры области внимания

1

1

Отчет по объектам привязки. Создание пометок, и ориентиров.

Конец потока(4) Привязка к ГСК

Оценка смещения, согласно движению наблюдателя

Оценка координат опорных объектов

Начало потока(4) Привязка к ГСК

**Этап 3. Анализ данных.**

Данные потока(1):

**Время.**

int timeStart = GetTickCount();

**Видеоданные: частота кадров, потоки RGB данных.**

Где яркосная составляющая:

Y =(B\*0.11 + G\*0.59 + R\*0.3);

//Инициализация из потока файла формата BMP-----------------------------------------------------

/\*BMP FILE HEADER

BM signature (2 bytes) =BM

File size (4 bytes)

Reserved (2 bytes)

Reserved (2 bytes)

Location of bitmap data (4 bytes)\*/

/\*INFORMATION HEADER

Size of information header (4 bytes) =40

n Image width (4 bytes)

m Image height (4 bytes)

Number of color planes (2 bytes) =1

Number of bits per pixel (2 bytes)

Compression method used (4 bytes)

Number of bytes of bitmap data (4 bytes)

Horizontal screen resolution (4 bytes)

Vertical screen resolution (4 bytes)

Number of colors used in the image (4 bytes) =0

Number of important colors (4 bytes) \*/

/\*virtual\*/image image::LoadFromFile(ifstream &file)

{

dm=dn=0;

unsigned char buf[1023];

file.read(buf, 54);

long int location = ((long int)(buf[10]))+((long int)(buf[11]) << 8)+((long int)(buf[12]) << 16)+((long int) (buf[13]) << 24);

int n = ((long int)(buf[18])) + ((long int)(buf[19]) << 8) + ((long int)(buf[20]) << 16) + ((long int)(buf[21]) << 24);

int m = ((long int)(buf[22]))+((long int)(buf[23]) << 8)+((long int)(buf[24]) << 16)+((long int) (buf[25]) << 24);

long int bytes\_bitmap\_data = ((long int)(buf[34]))+((long int)(buf[35]) << 8)+((long int)(buf[36]) << 16)+((long int) (buf[37]) << 24);

file.ignore((location - 54));

long int j = (m - 1), k = 0, i = 0, mem\_i = 0, s;

B = G = R = Y = Lnorm = Sx = Sy = Sxy = H = matric<float>(m, n);

while (mem\_i <= bytes\_bitmap\_data)

{

file.read(buf, 3);

while ((j >= 0) && (i <= 2))

{

B.ar[j\*B.n+k] = buf[i++];

G.ar[j\*G.n+k] = buf[i++];

R.ar[j\*R.n+k++] = buf[i++];

if (k >= n)

{

k = 0;

j--;

}

}

mem\_i+=i;

i = 0;

if (file.eof())

break;

}

file.close();

Y =(B\*0.11 + G\*0.59 + R\*0.3);

return \*this;

};

**Эквализация**

//Эквализация-----------------------------------------------------------------

/\*virtual\*/ image image :: Eqalization(void)

{

float Ymax=0, Ymin=255, Rmax=0, Rmin=255, Gmax=0, Gmin=255, Bmax=0, Bmin=255;

m = Y.m;

n = Y.n;

for (int j = 0; j < m; j++)

for (int k = 0; k < n; k++)

{

if(Y.ar[j\*n+k]>Ymax){Ymax=Y.ar[j\*n+k];}

if(Y.ar[j\*n+k]<Ymin){Ymin=Y.ar[j\*n+k];}

if(R.ar[j\*n+k]>Rmax){Rmax=R.ar[j\*n+k];}

if(R.ar[j\*n+k]<Rmin){Rmin=R.ar[j\*n+k];}

if(G.ar[j\*n+k]>Gmax){Gmax=G.ar[j\*n+k];}

if(G.ar[j\*n+k]<Gmin){Gmin=G.ar[j\*n+k];}

if(B.ar[j\*n+k]>Bmax){Bmax=B.ar[j\*n+k];}

if(B.ar[j\*n+k]<Bmin){Bmin=B.ar[j\*n+k];}

}

for (int j = 0; j < m; j++)

for (int k = 0; k < n; k++)

{

Y.ar[j\*n+k] = 255\*(Y.ar[j\*n+k]-Ymin)/(Ymax-Ymin);

R.ar[j\*n+k] = 255\*(R.ar[j\*n+k]-Rmin)/(Rmax-Rmin);

G.ar[j\*n+k] = 255\*(G.ar[j\*n+k]-Gmin)/(Gmax-Gmin);

B.ar[j\*n+k] = 255\*(B.ar[j\*n+k]-Bmin)/(Bmax-Bmin);

}

return \*this;

}

**Дискретное косинусное преобразование.**

\mathrm{DCT}\text{-}1_n= \left[\cos kl\tfrac{\pi}{n-1}\right]_{0\leq k,l<n} 

\mathrm{DCT}\text{-}2_n= \left[\cos k(l+\tfrac{1}{2})\tfrac{\pi}{n}\right]_{0\leq k,l<n} 

\mathrm{DCT}\text{-}3_n= \left[\cos (k+\tfrac{1}{2})l\tfrac{\pi}{n}\right]_{0\leq k,l<n} 

\mathrm{DCT}\text{-}4_n= \left[\cos (k+\tfrac{1}{2})(l+\tfrac{1}{2})\tfrac{\pi}{n}\right]_{0\leq k,l<n} 

// Discret Cosinus Transformation\*--------------------------------------------

matric<float> DCT(matric<float> &r, int M, int N, int NN)

{

matric<float> DCTkvant(NN, NN);

matric<float> kvant(NN, NN);

float PI = 3.1415926535897932384626433832795;

float Cu, Cv, AC;

for (int u = 0; u < NN; u++)

{

for (int v = 0; v < NN; v++)

{

if(u != 0)

{Cu=1;}

else

{Cu=1/(pow(2, 0.5));}

if(v != 0)

{Cv=1;}

else

{Cv=1/(pow(2, 0.5));}

AC=0;

for (int x = 0; x < NN; x++)

{

for (int y = 0; y < NN; y++)

{

if((y+M)<r.m && (x+N)<r.n) {AC += r.ar[(y+M)\*r.n+x+N]\*cos((2\*(x)+1)\*(u)\*PI/(2\*NN))\*cos((2\*(y)+1)\*(v)\*PI/(2\*NN));}

}

}

DCTkvant.ar[v\*NN+u]=0.25\*Cu\*Cv\*AC;

}

}

return DCTkvant;

}

// Inverse Discret Cosinus Transformation\*--------------------------------------------

matric<float> IDCT(matric<float> &r, int M, int N, int NN)

{

matric<float> DCTkvant(NN, NN);

matric<float> kvant(NN, NN);

float PI = 3.1415926535897932384626433832795;

float Cu, Cv, AC;

for (int u = 0; u < NN; u++)

{

for (int v = 0; v < NN; v++)

{

AC=0;

for (int x = 0; x < NN; x++)

{

for (int y = 0; y < NN; y++)

{

if(x != 0)

{Cu=1;}

else

{Cu=1/(pow(2, 0.5));}

if(y != 0)

{Cv=1;}

else

{Cv=1/(pow(2, 0.5));}

if((y+M)<r.m && (x+N)<r.n) {AC += Cu\*Cv\*r.ar[(y+M)\*r.n+x+N]\*cos((2\*(u)+1)\*(x)\*PI/(2\*NN))\*cos((2\*(v)+1)\*(y)\*PI/(2\*NN));}

}

}

DCTkvant.ar[v\*NN+u]=0.25\*AC;

}

}

**Преобразование Фурье**

Прямое преобразование:

X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2 \pi i}{N} k n} \qquad k = 0, \dots, N-1

Обратное преобразование:

x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} k n} \quad \quad n = 0,\dots,N-1.

Обозначения:

* *N* — количество значений сигнала, измеренных за период, а также количество компонентов разложения;
* x_n, \quad n = 0,\dots,N-1, — измеренные значения сигнала (в дискретных временных точках с номерами n = 0,\dots,N-1, которые являются входными данными для прямого преобразования и выходными для обратного;
* X_k, \quad k = 0,\dots,N-1, — *N* [комплексных](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) амплитуд синусоидальных сигналов, слагающих исходный сигнал; являются выходными данными для прямого преобразования и входными для обратного; поскольку амплитуды комплексные, то они обозначают одновременно и амплитуду и фазу;
* |X_k| \over N — обычная (вещественная) амплитуда k-го синусоидального сигнала;
* arg(*Xk*) — фаза k-го синусоидального сигнала ([аргумент комплексного числа](http://ru.wikipedia.org/wiki/%D0%90%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82_%D0%BA%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%BD%D0%BE%D0%B3%D0%BE_%D1%87%D0%B8%D1%81%D0%BB%D0%B0));
* *k* — частота k-го сигнала, равная \frac{k}{T}, где *T* — период времени, в течение которого брались входные данные.

Из последнего видно, что преобразование раскладывает сигнал на синусоидальные составляющие (которые называются гармониками) с частотами от N колебаний за период до одного колебания за период. Поскольку частота дискретизации сама по себе равна N отсчётов за период, то высокочастотные составляющие не могут быть корректно отображены — возникает [муаров эффект](http://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%B0%D1%80%D0%BE%D0%B2_%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82). Это приводит к тому, что первая половина из N комплексных амплитуд, фактически, является зеркальным отображением второй и не несёт значительного смысла

 \vec X = \hat A \vec x 


\hat A = \begin{pmatrix}
1 &1 &1 &1 &\ldots &1 \\
1 &e^{-\frac{2\pi i}{N}} &e^{-\frac{4\pi i}{N}} &e^{-\frac{6\pi i}{N}} &\ldots &e^{-\frac{2\pi i}{N}(N-1)}\\
1 &e^{-\frac{4\pi i}{N}} &e^{-\frac{8\pi i}{N}} &e^{-\frac{12\pi i}{N}} &\ldots &e^{-\frac{2\pi i}{N}2(N-1)}\\
1 &e^{-\frac{6\pi i}{N}} &e^{-\frac{12\pi i}{N}} &e^{-\frac{18\pi i}{N}} &\ldots &e^{-\frac{2\pi i}{N}3(N-1)}\\
\vdots &\vdots &\vdots &\vdots &\ddots &\vdots\\
1 &e^{-\frac{2\pi i}{N}(N-1)} &e^{-\frac{2\pi i}{N}2(N-1)} &e^{-\frac{2\pi i}{N}3(N-1)} &\ldots &e^{-\frac{2\pi i}{N}(N-1)^2}
\end{pmatrix}

 A(m,n) = \exp \left( -2\pi i \frac{(m-1)(n-1)}{N} \right) 

//Fast Furye Transformation --------------------------------------------------

matric<int> FFT(matric<float> &r, int M, int N)

{

matric<int> FFTmatric(8, 8);

float Sum1, Sum2;

for(int u=0; u<8; u++)

{

for (int v=0; v<8; v++)

{

Sum1=0;

for(int y=0; y<8; y++)

{

for(int x=0; x<8; x++)

{

Sum1 += exp(-2\*3.14\*u\*x/8) \* r.ar[(y+M)\*8+x+N] \* exp(-2\*3.14\*v\*y/8);

}

}

FFTmatric.ar[v\*8+u] = (int)(0.125\*0.125\*Sum1);

}

}

return FFTmatric;

}

 X_m=\sum_{n=0}^{M-1} x_{2n} a_{M}^{nm} + \exp \left( -2\pi i \frac{m}{N} \right) \sum_{n=0}^{M-1} x_{2n+1} a_{M}^{nm} 

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <stdio.h>

double \*FFT( int \*dIn, int nn )

{

int i, j, n, m, mmax, istep;

double tempr, tempi, wtemp, theta, wpr, wpi, wr, wi;

int isign = -1;

double \*data = new double [ nn \* 2 + 1 ];

for( i = 0; i < nn; i++ )

{

data[ i \* 2 ] = 0;

data[ i \* 2 + 1 ] = dIn[ i ];

}

n = nn << 1;

j = 1;

i = 1;

while( i < n )

{

if( j > i )

{

tempr = data[ i ]; data[ i ] = data[ j ]; data[ j ] = tempr;

tempr = data[ i + 1 ]; data[ i + 1 ] = data[ j + 1 ]; data[ j + 1 ] = tempr;

}

m = n >> 1;

while( ( m >= 2 ) && ( j > m ) )

{

j = j - m;

m = m >> 1;

}

j = j + m;

i = i + 2;

}

mmax = 2;

while( n > mmax )

{

istep = 2 \* mmax;

theta = 2.0 \* M\_PI / ( isign \* mmax );

wtemp = sin( 0.5 \* theta );

wpr = -2.0 \* wtemp \* wtemp;

wpi = sin( theta );

wr = 1.0;

wi = 0.0;

m = 1;

while( m < mmax )

{

i = m;

while( i < n )

{

j = i + mmax;

tempr = wr \* data[ j ] - wi \* data[ j + 1 ];

tempi = wr \* data[ j + 1 ] + wi \* data[ j ];

data[ j ] = data[ i ] - tempr;

data[ j + 1 ] = data[ i + 1 ] - tempi;

data[ i ] = data[ i ] + tempr;

data[ i + 1 ] = data[ i + 1 ] + tempi;

i = i + istep;

}

wtemp = wr;

wr = wtemp \* wpr - wi \* wpi + wr;

wi = wi \* wpr + wtemp \* wpi + wi;

m = m + 2;

}

mmax = istep;

}

double \*dOut = new double [ nn / 2 ];

for( i = 0; i < ( nn / 2 ); i++ )

{

dOut[ i ] = sqrt( data[ i \* 2 ] \* data[ i \* 2 ] + data[ i \* 2 + 1 ] \* data[ i \* 2 + 1 ] );

}

delete []data;

return dOut;

}

int main()

{

int \*dsin = new int [ 1801 ], \*pdsin = dsin;

for( double x = 0.; x <= 10. \* M\_PI; x += M\_PI / 180.0 )

{

\*pdsin++ = sin( x ) \* 1000. + cos( 0.5 \* x ) \* 1000.;

}

double \*dfourier = FFT( dsin, 1024 );

delete []dsin;

for( int i = 0; i < 512; i++ )

{

printf( "%d**\t**%lf**\r\n**", i, dfourier[ i ] );

}

delete []dfourier;

return 0;

}

**Матрицы фильтра Гаусса разного масштаба**

Создаем матрицы нормального распределения вероятности Гауса, с различными коэффициентами распределения: ;

//Фильтр Гаусса---------------------------------------------------------------

matric<float> FilterGauss (float sigma, int n, float factorY) // sigma(0.5..1.5) n(1..3) factorY(1..2.5)

{

matric<float> FilterGauss(n, n);

for (int j = 0; j < n; j++)

for (int k = 0; k < n; k++)

FilterGauss.ar[j\*n+k] = factorY \* (exp((-pow(k, 2)-pow(j, 2))/(2\*pow(sigma, 2)))/(2 \* M\_PI \* pow(sigma, 2)));

return FilterGauss;

};

float Etta = 1.2, S = 0.7, h;

scale = 1 .. 14;

GaussD = FilterGauss(S\*pow(Etta, scale), 2, 1);

**Свертка**

//Свертка---------------------------------------------------------------------

/\*virtual\*/image image :: Convolution(const matric<float> &Filter)

{

matric<float> A(Filter.m, Filter.n);

matric<float> temp(Y);

for (int jm = Filter.m; jm < Y.m; jm++)

{

for (int kn = Filter.n; kn < Y.n; kn++)

{

for (int j = 0; j < Filter.m; j++)

for (int k = 0; k < Filter.n; k++)

A.ar[j\*Filter.n+k] = temp.ar[(jm - Filter.m + j)\*Y.n+(kn - Filter.n + k)];

//Умножение A = A \* Filter и свертка в temp

for (int jj = 0; jj < A.m; jj++)

for (int kk = 0; kk < A.n; kk++)

{

int l = (jm - Filter.m + jj)\*Y.n + (kn - Filter.n + kk);

temp.ar[l] = 0;

for (int i = 0; i < A.n; i++)

temp.ar[l] += A.ar[jj\*A.n+i] \* Filter.ar[i\*Filter.n+kk];

}

//-------------------------------------

}

}

Y = temp;

return \*this;

};

**Нормализованный Лапласиан по масштабированным Гауссианам**

1. Сворачиваем битовые матрицы по фильтру Гаусса:
2. Находим Лапласиан Гауссиана: ;

В данном случае на изображение применяется 2D(двухмерный) фильтр Гаусса разного масштаба, и на каждый масштаб находится Лапласиан Гауссиана.

**Градиент**

|  |  |  |
| --- | --- | --- |
| **a** | **b** | **C** |
| **d** | **e** | **F** |
| **g** | **h** | **i** |

**Sx = c+2f+I – (a+2d+g)**

**Sy=g+2h+i-(a+2b+c)**

**S=**=

**Фактор Харрисона**

LoG(x, ) = , max по переменной n

Если LoG(x, ) < LoG(x, ) , LoG(x, LoG(x, ) и LoG(x,)>LoGthr>10, то этот масштаб наиболее информативен.

**Регрессия**

R = =





/\*virtual\*/ image image :: HarrissLaplass(void)

{

float Etta = 1.2, S = 0.7, h;

matric<float> A(2, 2), GaussN(2, 2), GaussD(2, 2);

int al, bl, cl, dl, el, fl, gl, hl, il, l, lx, ly, lxy;

int Ym = Y.m; int Yn = Y.n;

matric<float> L(Ym, Yn), Lx(Ym, Yn), Ly(Ym, Yn), Lxx(Ym, Yn), Lxy(Ym, Yn), Lxxyy(Ym, Yn), Lyy(Ym, Yn), LoG(Ym, Yn);

for(int scale = 1; scale < 10; scale++)

{

GaussN = FilterGauss(pow(Etta, scale), 2, 1);

GaussD = FilterGauss(S\*pow(Etta, scale), 2, 1);

for (int jm = GaussN.m+2; jm < Y.m; jm++)

{

for (int kn = GaussN.n+2; kn < Y.n; kn++)

{

for (int j = 0; j < GaussN.m; j++)

for (int k = 0; k < GaussN.n; k++)

A.ar[j\*GaussN.n+k] = Y.ar[(jm - GaussN.m + j)\*Y.n+(kn - GaussN.n + k)];

for (int jj = 0; jj < A.m; jj++)

for (int kk = 0; kk < A.n; kk++)

{

cl = (jm - GaussN.m + jj - 2)\*Y.n + (kn - GaussN.n + kk);

al = cl-2; // al = (jm - GaussN.m + jj - 2)\*Y.n + (kn - GaussN.n + kk - 2);

bl = cl-1; // bl = (jm - GaussN.m + jj - 2)\*Y.n + (kn - GaussN.n + kk - 1);

dl = (jm - GaussN.m + jj - 1)\*Y.n + (kn - GaussN.n + kk - 2);

lxy = el = (jm - GaussN.m + jj - 1)\*Y.n + (kn - GaussN.n + kk - 1);

ly = fl = (jm - GaussN.m + jj - 1)\*Y.n + (kn - GaussN.n + kk);

gl = (jm - GaussN.m + jj)\*L.n + (kn - GaussN.n + kk - 2);

lx = hl = (jm - GaussN.m + jj)\*Y.n + (kn - GaussN.n + kk - 1);

l = il = (jm - GaussN.m + jj)\*Y.n + (kn - GaussN.n + kk);

L.ar[l] = 0;

//Нормальный Лаплассиан Гауссиана

for (int i = 0; i < A.n; i++)

L.ar[l] += A.ar[jj\*A.n+i] \* (GaussD.ar[i\*GaussN.n+kk] - GaussN.ar[i\*GaussN.n+kk]);

Lnorm.ar[l] += L.ar[l] \* pow(S\*pow(Etta, scale), 2);

//Встроенный градиент

/\* Lx.ar[l] = (Lnorm.ar[l]-Lnorm.ar[lx]);

Ly.ar[l] = (Lnorm.ar[l]-Lnorm.ar[ly]);

Lxx.ar[l] = Lx.ar[l]-Lx.ar[lx];

Lyy.ar[l] = Lx.ar[l]-Lx.ar[ly];

Lxy.ar[l] = (Lnorm.ar[l] - Lnorm.ar[lxy]);

Lxxyy.ar[l] = Lxy.ar[l]-Lxy.ar[lxy];

if (scale == 1)

{

Sx.ar[l] = (Lnorm.ar[cl] + (2 \* Lnorm.ar[fl]) + Lnorm.ar[il]) - (Lnorm.ar[al] + (2 \* Lnorm.ar[dl]) + Lnorm.ar[gl]);

Sy.ar[l] = (Lnorm.ar[gl] + (2 \* Lnorm.ar[hl]) + Lnorm.ar[il]) - (Lnorm.ar[al] + (2 \* Lnorm.ar[bl]) + Lnorm.ar[cl]);

Sxy.ar[l] = pow(pow(Sx.ar[l], 2)+ pow(Sy.ar[l], 2), 0.5);

} \*/

// Фактор Харрисона

/\* if(pow(pow(Lxx.ar[l]+Lyy.ar[l], 2), 0.5)>LoG.ar[l] && pow(pow(Lxx.ar[l]+Lyy.ar[l], 2), 0.5)>10)

{

h = pow(Etta, scale)\*(Lxx.ar[l]\*Lyy.ar[l]) - Lxxyy.ar[l]\*Lxxyy.ar[l] +

0.04\*pow(pow(Etta, scale)\*(Lxx.ar[l]+Lyy.ar[l]), 2);

if(h > H.ar[l] && h>250)

{H.ar[l] = h/10;}

}

LoG.ar[l] = pow(pow(Lxx.ar[l]+Lyy.ar[l], 2), 0.5); \*/

}

}

}

}

return \*this;

}

**Набор характеристтик области внимания**

1. Стандартное отклонение (standart deviation)

среднее пиксельное значение сегмента

- пиксельное значение в I строке j столбце

1. Асимметрия (skewness)
2. Межпиксельная контрастность (neighbor contrast)

Средняя разность между значениями соседних пикселов

1. Бета (beta) показывает насколько сильно отличаются значения пикселов от значения в центре блока
2. Максимальный градиент (maximum gradient)

float \*mean, \*std\_dev, \*skewness, \*neighbor\_contrast, \*beta\_coeff;

mean = new float[KeyPoint];

std\_dev = new float[KeyPoint];

skewness= new float[KeyPoint];

neighbor\_contrast= new float[KeyPoint];

beta\_coeff= new float[KeyPoint];

for (int i = 0; i < KeyPoint; i++)

{

X1 = (int)v1[i].x - Segment;

Y1 = (int)v1[i].z - Segment;

X2 = (int)v1[i].x + Segment;

Y2 = (int)v1[i].z + Segment;

SumN = ((Y2+1)\*rimg.Y.n+(X2+1) - Y1\*rimg.Y.n+X1);

if(SumN)

{

center\_row = (Y2+1 - Y1)/2 + Y1;

center\_col = (X2+1 - X1)/2 + X1;

for (int j = Y1; j < Y2+1; j++)

{

row\_dist\_sq = (j-center\_row)\*(j-center\_row);

for (int k = X1; k < X2+1; k++)

{

col\_dist = k-center\_col;

mean[i] += rimg.Y.ar[j\*rimg.Y.n+k];

mean\_rad += sqrt(row\_dist\_sq + col\_dist\*col\_dist);

}

}

mean[i] = mean[i]/SumN;

mean\_rad /= SumN;

for (int j = Y1; j < Y2+1; j++)

{

for (int k = X1; k < X2+1; k++)

{

x = mean\_rad - sqrt(row\_dist\_sq + col\_dist\*col\_dist);

sxy += x\*(rimg.Y.ar[j\*rimg.Y.n+k] - mean[i]);

sxx += x\*x;

diff = rimg.Y.ar[j\*rimg.Y.n+k]-mean[i];

std\_dev[i] += diff\*diff;

skewness[i] += diff\*diff\*diff;

}

}

for (int j = Y1+incr; j < Y2+1; j++)

for (int k = X1+incr; k < X2+1; k++)

neighbor\_contrast[i] += (float)abs((int)((rimg.Y.ar[j\*rimg.Y.n+k] - rimg.Y.ar[(j-incr)\*rimg.Y.n+k])\*100))/100 + (float)abs((int)((rimg.Y.ar[j\*rimg.Y.n+k] - rimg.Y.ar[j\*rimg.Y.n+(k-incr)])\*100))/100;

if(std\_dev[i] < 1e-10 && (-std\_dev[i]) < 1e-6){std\_dev[i] =0.0;}

else{std\_dev[i] = sqrt(std\_dev[i]/SumN);}

skewness[i] /= SumN;

skewness[i] = skewness[i]/(std\_dev[i]\*std\_dev[i]\*std\_dev[i]);

neighbor\_contrast[i] /= SumN;

if (sxx < 1e-10) {beta\_coeff[i] = 0.0;}

else {beta\_coeff[i] = sxy/sxx;}

}

if(i==0)

{

meanMIN = meanMAX = mean[i];

std\_devMIN = std\_devMAX = std\_dev[i];

skewnessMIN = skewnessMAX = skewness[i];

neighbor\_contrastMIN = neighbor\_contrastMAX = neighbor\_contrast[i];

beta\_coeffMIN = beta\_coeffMAX = beta\_coeff[i];

}

if(meanMIN > mean[i])meanMIN = mean[i];

if(meanMAX < mean[i])meanMAX = mean[i];

if(std\_devMIN > std\_dev[i])std\_devMIN = std\_dev[i];

if(std\_devMAX < std\_dev[i])std\_devMAX = std\_dev[i];

if(skewnessMIN > skewness[i])skewnessMIN = skewness[i];

if(skewnessMAX < skewness[i])skewnessMAX = skewness[i];

if(neighbor\_contrastMIN > neighbor\_contrast[i])neighbor\_contrastMIN = neighbor\_contrast[i];

if(neighbor\_contrastMAX < neighbor\_contrast[i])neighbor\_contrastMAX = neighbor\_contrast[i];

if(beta\_coeffMIN > beta\_coeff[i])beta\_coeffMIN = beta\_coeff[i];

if(beta\_coeffMAX < beta\_coeff[i])beta\_coeffMAX = beta\_coeff[i];

}

for (int i = 0; i < KeyPoint; i++)

{

mean[i] = (mean[i]-meanMIN)/(meanMAX-meanMIN);

std\_dev[i] = (std\_dev[i]-std\_devMIN)/(std\_devMAX-std\_devMIN);

skewness[i] = (skewness[i]-skewnessMIN)/(skewnessMAX-skewnessMIN);

neighbor\_contrast[i] = (neighbor\_contrast[i]-neighbor\_contrastMIN)/(neighbor\_contrastMAX-neighbor\_contrastMIN);

beta\_coeff[i] = (beta\_coeff[i]-beta\_coeffMIN)/(beta\_coeffMAX-beta\_coeffMIN);

}

delete mean, std\_dev, skewness, neighbor\_contrast, beta\_coeff;

Данные потока(2):

**Аффинное преобразование**

\begin{pmatrix} f(x) \\ 1 \end{pmatrix} = \begin{pmatrix} M & v \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix}

//Аффинное преобразование-----------------------------------------------------

void AffinTransformation(image &img, matric<float> &affin)

{

float a, b, c, d, e, f, jaff, kaff;

a = affin.ar[0\*3+0];

b = affin.ar[0\*3+1];

e = affin.ar[0\*3+2];

c = affin.ar[1\*3+0];

d = affin.ar[1\*3+1];

f = affin.ar[1\*3+2];

int m, n, x[4], y[4], Xmin=0, Ymin=0, Xmax=0, Ymax=0;

x[0]=(-e)/a;

y[0]=(-f)/d;

x[1]=(-b\*(img.R.m)-e)/a;

y[1]=((img.R.m)-f)/d;

x[2]=((img.R.n)-e)/a;

y[2]=(-c\*(img.R.n)-f)/d;

x[3]=((img.R.n)-b\*(img.R.m)-e)/a;

y[3]=((img.R.m)-c\*(img.R.n)-f)/d;

for(int i=0; i<4; i++)

{

if(Xmin>x[i]){Xmin=x[i];}

if(Ymin>y[i]){Ymin=y[i];}

if(Xmax<x[i]){Xmax=x[i];}

if(Ymax<y[i]){Ymax=y[i];}

}

m = (int)(Ymax-Ymin);

n = (int)(Xmax-Xmin);

Form1->Image1->Canvas->Rectangle(0,0,800,600);

for (int j = 0; j <800; j++)

for (int k = 0; k < 600; k++)

{

jaff =c\*j + d\*k + f;

kaff =a\*j + b\*k + e;

if(kaff>0 && jaff>0 && kaff<img.R.n && jaff<img.R.m)

{

Form1->Image1->Canvas->Pixels[(int)j][(int)k] = (int)img.R.ar[(int)jaff\*img.R.n + (int)kaff] + ((int)(img.G.ar[(int)jaff\*img.R.n + (int)kaff]) << 8) + ((int)(img.B.ar[(int)jaff\*img.R.n + (int)kaff]) << 16);

}

}

};

**Определение матрицы аффинного преобразования**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Form1->Memo1->Lines->Add("Хороших точек: " + (String)(goodPoints[0]-2));

delta = deltaA = deltaB = deltaC = deltaD = deltaE = deltaF = delta \*0;

for (int i = 1; i < (goodPoints[0]-2); i++)

{

delta.ar[0\*delta.n+0] += v1[goodPoints[i]].x;

delta.ar[1\*delta.n+0] += v1[goodPoints[i+1]].x;

delta.ar[2\*delta.n+0] += v1[goodPoints[i+2]].x;

delta.ar[0\*delta.n+1] += v1[goodPoints[i]].z;

delta.ar[1\*delta.n+1] += v1[goodPoints[i+1]].z;

delta.ar[2\*delta.n+1] += v1[goodPoints[i+2]].z;

delta.ar[0\*delta.n+2] = delta.ar[1\*delta.n+2] = delta.ar[2\*delta.n+2] += 1;

deltaA.ar[0\*deltaA.n+0] += v2[goodPoints[i]].x;

deltaA.ar[1\*deltaA.n+0] += v2[goodPoints[i+1]].x;

deltaA.ar[2\*deltaA.n+0] += v2[goodPoints[i+2]].x;

deltaA.ar[0\*deltaA.n+1] += v1[goodPoints[i]].z;

deltaA.ar[1\*deltaA.n+1] += v1[goodPoints[i+1]].z;

deltaA.ar[2\*deltaA.n+1] += v1[goodPoints[i+2]].z;

deltaA.ar[0\*deltaA.n+2] = deltaA.ar[1\*deltaA.n+2] = deltaA.ar[2\*deltaA.n+2] += 1;

deltaB.ar[0\*deltaB.n+0] += v1[goodPoints[i]].x;

deltaB.ar[1\*deltaB.n+0] += v1[goodPoints[i+1]].x;

deltaB.ar[2\*deltaB.n+0] += v1[goodPoints[i+2]].x;

deltaB.ar[0\*deltaB.n+1] += v2[goodPoints[i]].x;

deltaB.ar[1\*deltaB.n+1] += v2[goodPoints[i+1]].x;

deltaB.ar[2\*deltaB.n+1] += v2[goodPoints[i+2]].x;

deltaB.ar[0\*deltaB.n+2] = deltaB.ar[1\*deltaB.n+2] = deltaB.ar[2\*deltaB.n+2] += 1;

deltaE.ar[0\*deltaE.n+0] += v1[goodPoints[i]].x;

deltaE.ar[1\*deltaE.n+0] += v1[goodPoints[i+1]].x;

deltaE.ar[2\*deltaE.n+0] += v1[goodPoints[i+2]].x;

deltaE.ar[0\*deltaE.n+1] += v1[goodPoints[i]].z;

deltaE.ar[1\*deltaE.n+1] += v1[goodPoints[i+1]].z;

deltaE.ar[2\*deltaE.n+1] += v1[goodPoints[i+2]].z;

deltaE.ar[0\*deltaE.n+2] += v2[goodPoints[i]].x;

deltaE.ar[1\*deltaE.n+2] += v2[goodPoints[i+1]].x;

deltaE.ar[2\*deltaE.n+2] += v2[goodPoints[i+2]].x;

deltaC.ar[0\*deltaC.n+0] += v2[goodPoints[i]].z;

deltaC.ar[1\*deltaC.n+0] += v2[goodPoints[i+1]].z;

deltaC.ar[2\*deltaC.n+0] += v2[goodPoints[i+2]].z;

deltaC.ar[0\*deltaC.n+1] += v1[goodPoints[i]].z;

deltaC.ar[1\*deltaC.n+1] += v1[goodPoints[i+1]].z;

deltaC.ar[2\*deltaC.n+1] += v1[goodPoints[i+2]].z;

deltaC.ar[0\*deltaC.n+2] = deltaC.ar[1\*deltaC.n+2] = deltaC.ar[2\*deltaC.n+2] += 1;

deltaD.ar[0\*deltaD.n+0] += v1[goodPoints[i]].x;

deltaD.ar[1\*deltaD.n+0] += v1[goodPoints[i+1]].x;

deltaD.ar[2\*deltaD.n+0] += v1[goodPoints[i+2]].x;

deltaD.ar[0\*deltaD.n+1] += v2[goodPoints[i]].z;

deltaD.ar[1\*deltaD.n+1] += v2[goodPoints[i+1]].z;

deltaD.ar[2\*deltaD.n+1] += v2[goodPoints[i+2]].z;

deltaD.ar[0\*deltaD.n+2] = deltaD.ar[1\*deltaD.n+2] = deltaD.ar[2\*deltaD.n+2] += 1;

deltaF.ar[0\*deltaF.n+0] += v1[goodPoints[i]].x;

deltaF.ar[1\*deltaF.n+0] += v1[goodPoints[i+1]].x;

deltaF.ar[2\*deltaF.n+0] += v1[goodPoints[i+2]].x;

deltaF.ar[0\*deltaF.n+1] += v1[goodPoints[i]].z;

deltaF.ar[1\*deltaF.n+1] += v1[goodPoints[i+1]].z;

deltaF.ar[2\*deltaF.n+1] += v1[goodPoints[i+2]].z;

deltaF.ar[0\*deltaF.n+2] += v2[goodPoints[i]].z;

deltaF.ar[1\*deltaF.n+2] += v2[goodPoints[i+1]].z;

deltaF.ar[2\*deltaF.n+2] += v2[goodPoints[i+2]].z;

//вывод векторного поля

Form1->Image2->Canvas->Refresh();

Form1->Image2->Canvas->MoveTo(v2[goodPoints[i]].x, v2[goodPoints[i]].z);

switch (ColorFlag){

case 0: Form1->Image2->Canvas->Pen->Color = clRed; break;

case 1: Form1->Image2->Canvas->Pen->Color = clGreen; break;

case 2: Form1->Image2->Canvas->Pen->Color = clBlue; break;

case 3: Form1->Image2->Canvas->Pen->Color = clYellow; break; }

Form1->Image2->Canvas->LineTo(v1[goodPoints[i]].x, v1[goodPoints[i]].z);

}

if(delta.D()>(1e-15) || -delta.D()>(1e-15))

{

AffinTransformation.ar[0\*AffinTransformation.n+0] = deltaA.D()/delta.D();

AffinTransformation.ar[0\*AffinTransformation.n+1] = deltaB.D()/delta.D();

AffinTransformation.ar[1\*AffinTransformation.n+0] = deltaC.D()/delta.D();

AffinTransformation.ar[1\*AffinTransformation.n+1] = deltaD.D()/delta.D();

AffinTransformation.ar[0\*AffinTransformation.n+2] = deltaE.D()/delta.D();

AffinTransformation.ar[1\*AffinTransformation.n+2] = deltaF.D()/delta.D();

}

else

{

AffinTransformation.ar[0\*AffinTransformation.n+0] = 1.0;

AffinTransformation.ar[0\*AffinTransformation.n+1] = 0;

AffinTransformation.ar[1\*AffinTransformation.n+0] = 0;

AffinTransformation.ar[1\*AffinTransformation.n+1] = 1.0;

AffinTransformation.ar[0\*AffinTransformation.n+2] = 0;

AffinTransformation.ar[1\*AffinTransformation.n+2] = 0;

}

delete d, v, v1, v2, v3;

return AffinTransformation;

**Определение общего нарастающего изменения**

Memo1->Lines->Add("Прошло времени:");

Memo1->Lines->Add((GetTickCount() - timeStart)/1000);

for(int i = 1; i < (file\_collection); i++)

{

StatusBar1->SimpleText = "Обработка изображения № " + (String)(i+1);

F[1] = Glue(F[1], F[i+1], affin[1]);

Memo1->Lines->Add("Глобальные аффинные координаты смещения изображений " + (String)i + " и " + (String)(i+1));

Memo1->Lines->Add(affin[1].Show());

affin[1].ar[0] = affin[1].ar[0] \* affin[i+1].ar[0] + affin[1].ar[1] \* affin[i+1].ar[3];

affin[1].ar[1] = affin[1].ar[0] \* affin[i+1].ar[1] + affin[1].ar[1] \* affin[i+1].ar[4];

affin[1].ar[2] = affin[1].ar[0] \* affin[i+1].ar[2] + affin[1].ar[1] \* affin[i+1].ar[5] + affin[1].ar[2];

affin[1].ar[3] = affin[1].ar[3] \* affin[i+1].ar[0] + affin[1].ar[4] \* affin[i+1].ar[3];

affin[1].ar[4] = affin[1].ar[3] \* affin[i+1].ar[1] + affin[1].ar[4] \* affin[i+1].ar[4];

affin[1].ar[5] = affin[1].ar[3] \* affin[i+1].ar[2] + affin[1].ar[4] \* affin[i+1].ar[5] + affin[1].ar[5];

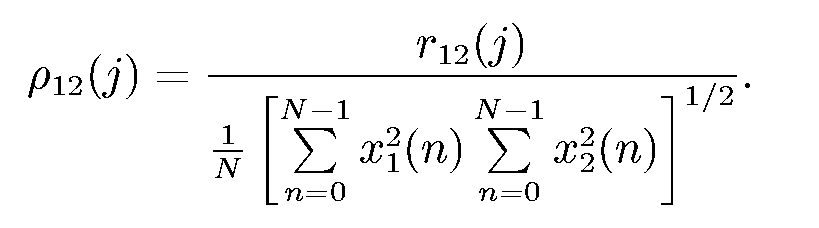
}

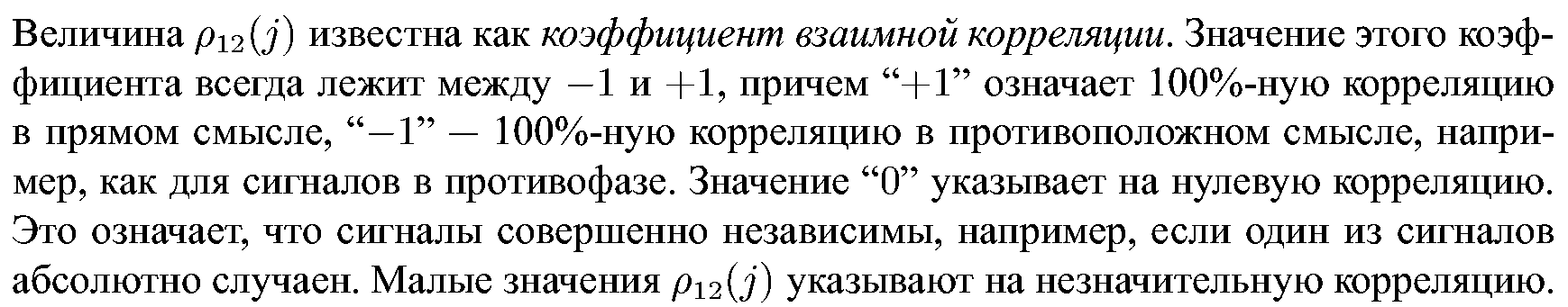
StatusBar1->SimpleText = "Вывод на экран склеенной матрицы.";

F[1].Draw();

Данные потока(3):

**Корреляция**





//Среднее значение------------------------------------------------------------

long double X(matric<float> &r, int M, int N, float R)

{

long double X = 0, XX = 0;

long int Sum = 0, Sum2 = 0;

for (int j = (M-R/2); j < (M+R/2); j++)

{

for (int k = (N-R/2); k < (N+R/2); k++)

if((pow(j-M, 2) + pow(k-N, 2) - pow(R, 2)) < 1)

{

X += r.ar[j\*r.n+k];

Sum++;

}

XX += X / Sum;

Sum2++;

Sum = 0;

X = 0;

}

XX = XX / Sum2;

return XX;

};

//Мат ожидание ---------------------------------------------------------------

long double Mmn(matric<float> &r, int M, int N, float R)

{

long int Sum = 0;

long double x, Mmn = 0;

x = X(r, M, N, R);

for (int j = (M-R/2); j < (M+R/2); j++)

for (int k = (N-R/2); k < (N+R/2); k++)

if((pow(j-M, 2) + pow(k-N, 2) - pow(R, 2)) < 1)

{

Mmn += r.ar[j\*r.n+k]\*x;

Sum++;

}

Mmn = Mmn/(Sum-1);

return Mmn;

};

//Второй момент ---------------------------------------------------------------

long double DXmn(matric<float> &r, int M, int N, float R)

{

long double Fm, x, D = 0;

long int Sum = 0;

x = X(r, M, N, R);

Fm = Mmn(r, M, N, R);

for (int j = (M-R/2); j < (M+R/2); j++)

for (int k = (N-R/2); k < (N+R/2); k++)

if((pow(j-M, 2) + pow(k-N, 2) - pow(R, 2)) < 1)

{

D += r.ar[j\*r.n+k]\*Fm\*x;

Sum++;

}

D = D/(Sum-1);

return D;

};

//Регрессия -------------------------------------------------------------------

long double regres(matric<float> &r1, matric<float> &r2, int r1M, int r1N, int r2M, int r2N, float R)

{

long double Regres = 0;

Regres = pow(pow(DXmn(r1, r1M, r1N, R)-DXmn(r2, r2M, r2N, R), 2), 0.5);

return Regres;

}

//Плоская корреляция-----------------------------------------------------------

long double Jxy(matric<float> &r1, matric<float> &r2, int r1M, int r1N, int r2M, int r2N, float R)

{

long double J = 0;

for (int j = (r1M-R/2); j < (r1M+R/2); j++)

for (int k = (r1N-R/2); k < (r1N+R/2); k++)

if((pow(j-r1M, 2) + pow(k-r1N, 2) - pow(R, 2)) < 1)

{

J += r1.ar[j\*r1.n+k]\*(j+k) - r2.ar[(j+r2M-r1M)\*r2.n+(k+r2N-r1N)]\*((j+r2M-r1M)+(k+r2N-r1N));

}

J = pow(J\*J, 0.5);

return J;

}