маСанкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт о лабораторной работе №4**

**Дисциплина:** Методы и средства цифровой обработки информации

Выполнил студент гр. 13541/1 А.А. Дроздовский

(подпись)

Руководитель Н.А. Абрамов

(подпись)

« » 2017 г.

Санкт – Петербург

2017

**СОДЕРЖАНИЕ**

[**1.** **Цель работы** 3](#_Toc496848237)

[**2.** **Ход работы** 3](#_Toc496848238)

[**2.1.** **Фильтр Гаусса** **Ошибка! Закладка не определена.**](#_Toc496848239)

[**2.2.** **Оператор Лапласиана** **Ошибка! Закладка не определена.**](#_Toc496848240)

[**2.3.** **Повышение резкости** **Ошибка! Закладка не определена.**](#_Toc496848241)

[**3.** **Выводы** 9](#_Toc496848242)

[***ПРИЛОЖЕНИЕ*** 10](#_Toc496848243)

[Текст программы 10](#_Toc496848244)

1. **Цель работы**

Изучить алгоритм поиска векторов смещения на последовательных кадрах.

Восстановить предшествующий кадр, зная вектора смещения.

1. **Ход работы**

Алгоритм работы:

1. Считать два последовательных кадра t-1 и t;
2. Изображение t-1 разбить на блоки размера Block с шагом Step;
3. Для каждого блока найти область поиска на изображении t, размером Search\_Area\*2 Х Search\_Area\*2;
4. В области поиска найти блок с наименьшим SAD;
5. Построить вектор смещения;
6. Восстановить кадр t-1.

*Pxt-1, Pxt* – оттенок пикселя ( с координатами i,j ) в изображениях t-1 и t соответственно.

Ниже представлены два последовательных кадра:



Рис. 2.1. Кадр t-1.

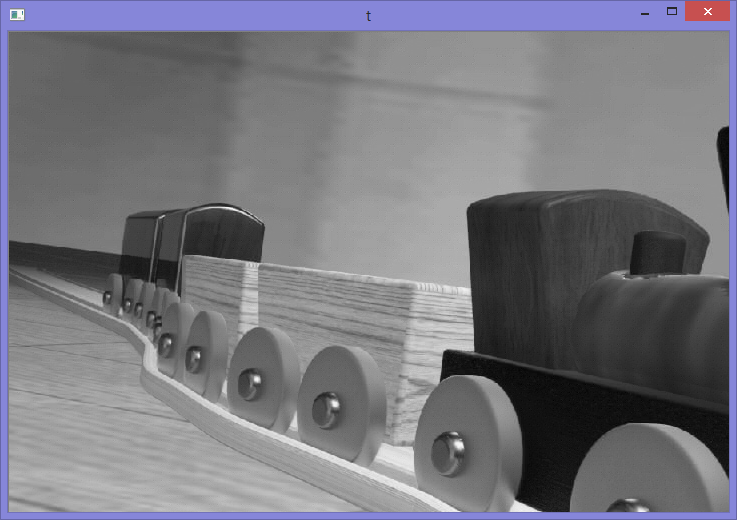


Рис. 2.2. Кадр t.

* 1. **Block = Block= 9, Step = 9, Search\_Area = 25**

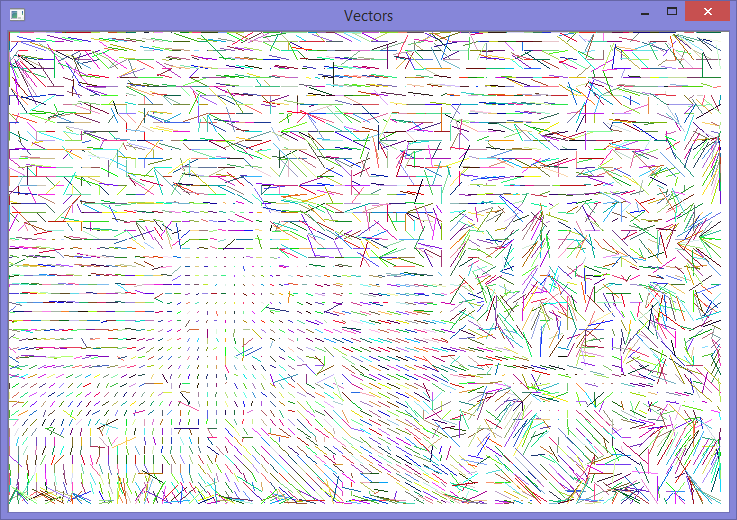


Рис. 2.3. Вектора смещения.



Рис. 2.4. Восстановленный кадр.

* 1. **Block= 9, Step = 4 Search\_Area = 25**

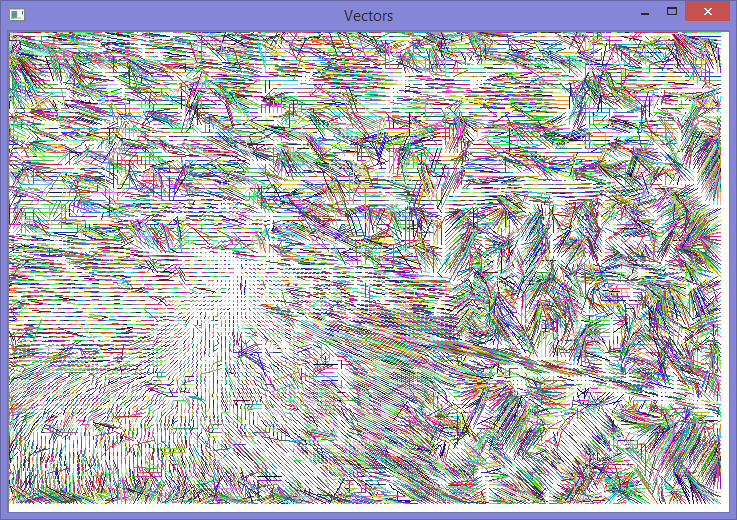


Рис. 2.5. Вектора смещения.

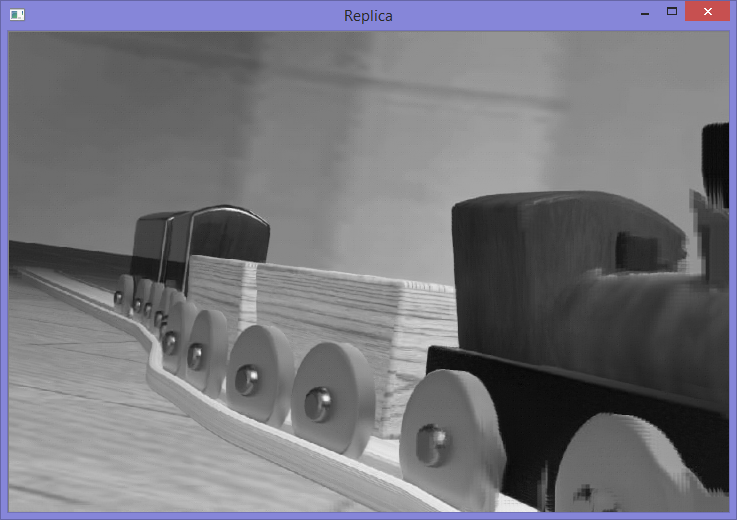


Рис. 2.6. Восстановленный кадр.

* 1. **Block= 5, Step = 5, Search\_Area = 25**

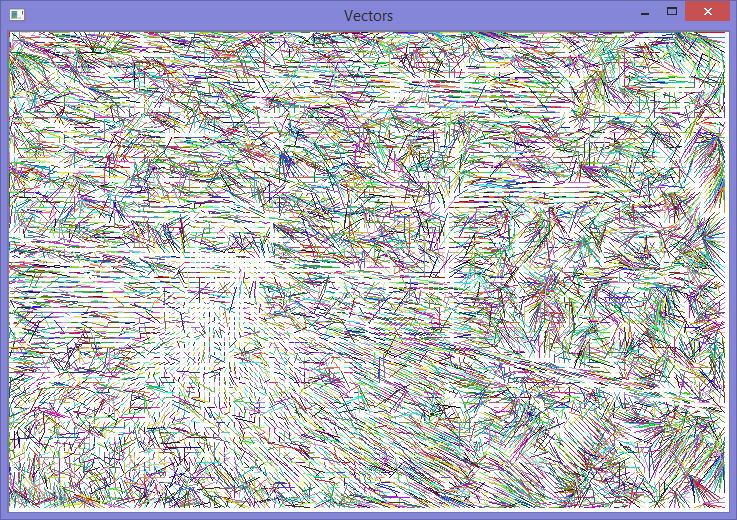


Рис. 2.7. Вектора смещения.

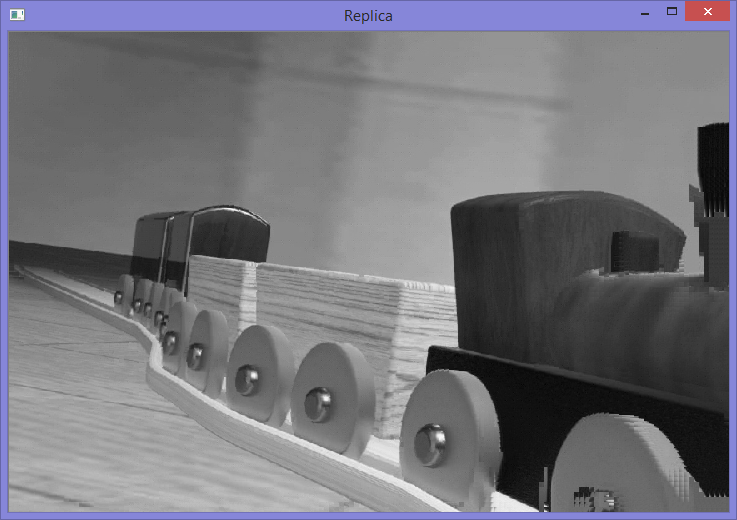


Рис. 2.8. Восстановленный кадр.

* 1. **Block = 5, Step = 2, Search\_Area = 25**

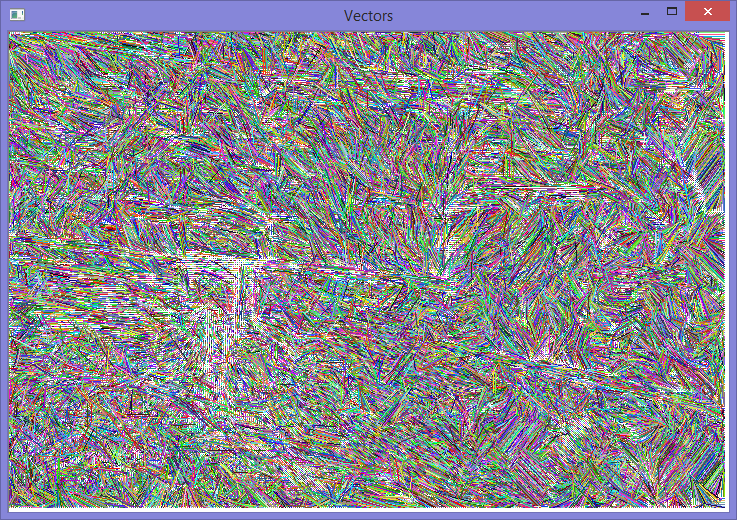


Рис. 2.9. Вектора смещения.

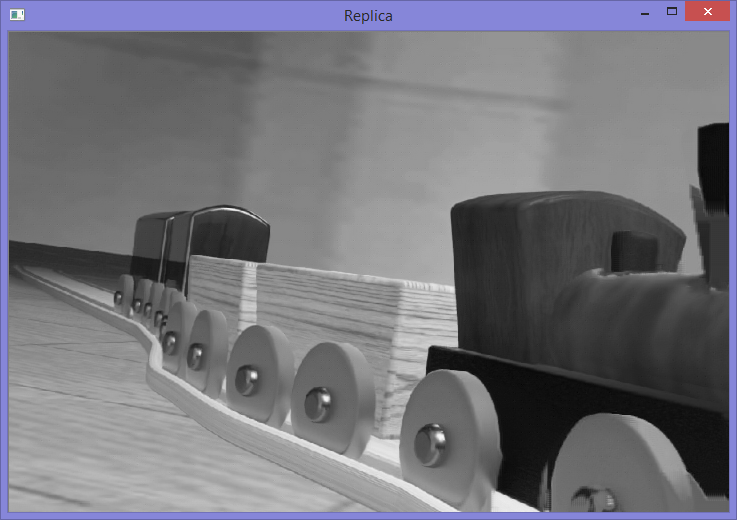
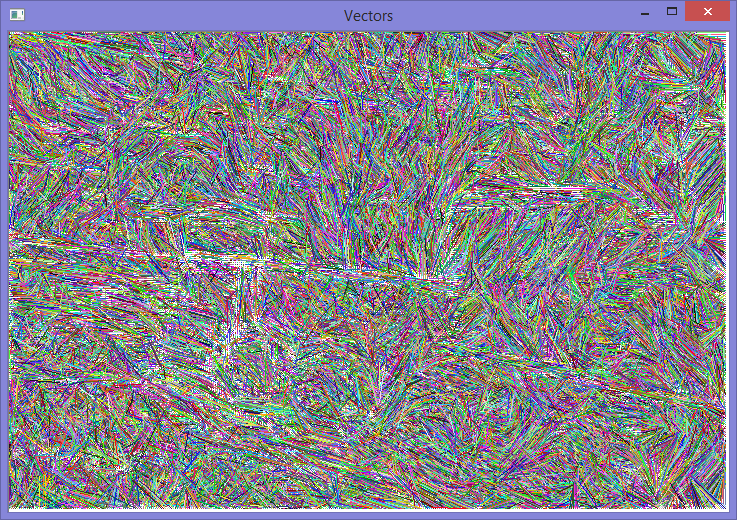


Рис. 2.10. Восстановленный кадр.

* 1. **Block = 4, Step = 2, Search\_Area = 32**

  
Рис. 2.11. Вектора смещения.

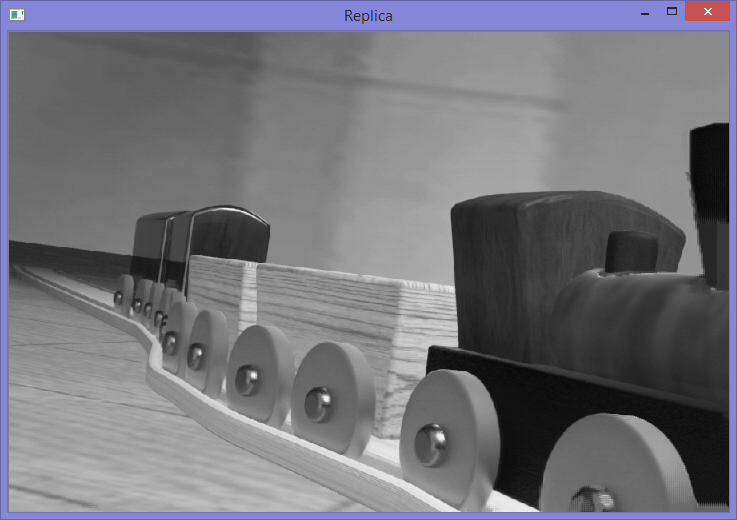


Рис. 2.12. Восстановленный кадр.

1. **Выводы**

Из результатов видно, что алгоритм, применяемый в данной работе, полностью рабочий, однако качество выходного изображения сильно зависит от параметров, а именно от размера блока (Block), шага блоков(Step), размера поискового окна(Search\_Area). Чем больше размер окна для поиска, тем выше шанс найти наиболее подходящее смещение, но при его увеличении значительно уменьшается быстродействие (при опыте в пункте 2.5. полное выполнение алгоритма заняло ~15 минут). Чем меньше шаг и размер блока, тем точнее выходной результат.

Для улучшения качества выходного изображения, при наложении двух и более пикселей нужно с весом SAD (чем меньше SAD, тем выше вероятность совпадения) высчитать выходное значения пикселя. Но из-за трудоемкости и для упрощения алгоритма в работе значения SAD никак не влияет на конечный оттенок пикселя. В качестве оттенка записывается среднее арифметическое из восстановленных в пиксель значений.

# ***ПРИЛОЖЕНИЕ***

## Текст программы

#define Block 4

#define Search\_Area 32

#define Step 2

#define \_SAD true

#define \_SSD false

using namespace cv;

using namespace std;

template <typename \_Tp>

Mat\_<uchar> Convert(Mat\_<\_Tp> input);

template <typename \_Tp>

\_Block Search\_Block(Mat\_<\_Tp> t\_1, Mat\_<\_Tp> t, int row, int col,int count\_row, int count\_col, int i\_old, int j\_old);

template <typename \_Tp>

void White(Mat\_<\_Tp> &input);

int main()

{

namedWindow("LR4");

String filename("LR4.avi"); // Имя видеофрагмента

VideoCapture capture = VideoCapture(filename);

Mat\_<uchar> frame, // Для считывания кадра

t, t\_1; // Два последовательных кадра

vector<\_Block> Blocks\_info; // Массив содержащий информацию про блоки

vector<Mat\_<uchar>> Blocks // Массив для хранения блоков размера Block x Block

,Search\_win; // Массив для хранения окна для поиска

// Считывание первого кадра

capture >> frame;

cvtColor(frame, t\_1, CV\_BGR2GRAY );

imshow("t\_1", t\_1);

// Считываем второго кадр

capture >> frame;

cvtColor(frame, t, CV\_BGR2GRAY);

imshow("t", t);

Mat\_<Vec3b> vector // Изображение для отрисовки векторов

= Mat\_<Vec3b>(t.rows, t.cols, Vec3b(255, 255, 255));

Mat\_<int> replica = // Для накапливания оттенков востановленного кадра

Mat\_<int>(t\_1.rows, t\_1.cols);

Mat\_<uchar> accumulator = // Аккумулятор количества перезаписи значения пикселя

Mat\_<uchar>(t\_1.rows, t\_1.cols);

White(replica);

White(accumulator);

// Формируем 2 массива из кадра t\_1 и t:

// 1 - содержащий блоки размером Block x Block

// 2 - содержащий окная для поиска

for (int row = 0; row < t\_1.rows; row += Step)

{

if (t\_1.rows - row < Block)

row = t\_1.rows - Block;

// формирование окна для поиска по вертикали

int i\_window = max(0, row - Search\_Area);

int count\_row = min(row - i\_window,Search\_Area)

+min(Search\_Area,t\_1.rows-row);

for (int col = 0; col < t\_1.cols; col += Step)

{

if (t\_1.cols - col < Block)

col=t\_1.cols - Block;

// формирование окна для поиска по горизонтали

int j\_window = max(0,col-Search\_Area);

int count\_col = min(col - j\_window, Search\_Area)

+ min(Search\_Area, t\_1.cols - col);

// Запись в массив информации о текущем блоке

\_Block bl = Search\_Block(t\_1, t, i\_window, j\_window, count\_row, count\_col, row, col);

Blocks\_info.push\_back(bl);

// Отрисовка векторов:

int RED = rand() % 256;

int GREEN = rand() % 256;

int BLUE = rand() % 256;

line(vector, Point(bl.ret\_j\_old(),bl.ret\_i\_old()),

Point(bl.ret\_j\_new(), bl.ret\_i\_new()),

Scalar(RED, GREEN, BLUE), 1, 8, 0);

// Восстановление изображения

for (int \_row = 0; \_row < Block; \_row++)

for (int \_col = 0; \_col < Block; \_col++)

{

replica.at<int>(bl.ret\_i\_old() + \_row, bl.ret\_j\_old() + \_col)

+= t.at<uchar>(bl.ret\_i\_new() + \_row, bl.ret\_j\_new() + \_col);

accumulator.at<uchar>(bl.ret\_i\_old() + \_row, bl.ret\_j\_old() + \_col) += 1;

}

if (col == t\_1.cols - Block)

break;

}

if (row == t\_1.rows - Block)

break;

}

// Подготовка восстановленного изображения к выводу на экран

Mat\_<uchar> rep =

Mat\_<uchar>(replica.rows, replica.cols);

for (int i = 0; i < replica.rows\*replica.cols; i++)

rep.at<uchar>(i) = round(replica.at<int>(i) / accumulator.at<uchar>(i));

imshow("Vectors", vector);

imshow("Replica", rep);

cv::waitKey(0);

return 0;

}

/\*\*

\* Преобразование Mat\_<typename> в Mat\_<uchar>

\*

\* Исходная матрица изображения нормализуется

в формат оттенков пикселей от 0 до 255.

Затем выводится в качестве выходного параметра.

\* @input Mat\_<typename> исходное изображение

\* @return Mat\_<uchar>

\*/

template <typename \_Tp>

Mat\_<uchar> Convert(Mat\_<\_Tp> input)

{

Mat\_<uchar> output(input.rows, input.cols);

normalize(input, input, 0, // Линейное растяжение

255, NORM\_MINMAX, -1, Mat());

output = input;

return output;

}

/\*\*

\* Поиска подходящего блока

\*

\* В окне для поиска находится блок с наименьшим

значением SAD. Формируется переменная класса \_Block

которая содержит координаты исходного блока,

координаты подходящего блока,

найденный SAD.

Переменная выводится в качестве выходного параметра

\* @t\_1 - Mat\_<typename> - исходное изображение,

содержащее блок для поиска

\* @t - Mat\_<typename> - исходное изображение,

содержащее окно для поиска

\* @row - int - начальная координата окная для поиска по вертикали

\* @col - int - начальная координата окная для поиска по горизонтали

\* @count\_row, @count\_col - int - величина окна для поиска

\* @i\_old, @j\_old - int - координаты текущего блока

\* @return \_Block - информация по текущему блоку

\*/

template <typename \_Tp>

\_Block Search\_Block(Mat\_<\_Tp> t\_1, Mat\_<\_Tp> t, int row, int col, int count\_row, int count\_col, int i\_old, int j\_old)

{

\_Block bl = \_Block(i\_old, j\_old);

int SAD = 0;

int SSD = 0;

for (int i = row; i < row + count\_row - Block; i++)

for (int j = col; j < col + count\_col - Block; j++)

{

SAD = 0;

for (int ii = 0; ii < Block; ii++)

{

for (int jj = 0; jj < Block; jj++)

SAD += abs(t\_1.at<\_Tp>(i\_old + ii, j\_old + jj) - t.at<\_Tp>(i + ii, j + jj));

if (bl.ret\_SAD() < SAD)

break;

}

if (\_SAD)

{

if (bl.ret\_SAD() > SAD)

{

bl.reload(i, j, SAD, 9999);

}

}

}

return bl;

}

/\*\*

\* Заливка изображения черным цветом

\*

\* Исходная матрица изображения заполняется

черным цветом (оттенком с индексом 0)

\* @input Mat\_<typename> исходное изображение

\*/

template <typename \_Tp>

void White(Mat\_<\_Tp> &input)

{

for (int i = 0; i < input.rows\*input.cols; i++)

input.at<\_Tp>(i) = 0;

}