САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет
По лабораторной работе №1
Дисциплина: Разработка графических приложений

Выполнил студент группы: 13541/3: Покатило П.А.

Преподаватель: Абрамов Н.А.

Санкт-Петербург
2018

Задание

Разработать программу, осуществляющую обнаружение сдвигов между двумя изображениями с помощью библиотеки OpenCV. Использовать метод полного перебора и метрику SAD.

Ход работы

Метрика SAD (Sum of absolute differences) – представляет из себя сумму модулей разностей всех элементов некоторой матрицы:

$$SAD = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |C_{ij} - R_{ij}|$$

Данную метрику реализует функция `bool compareSAD(cv::Mat src, cv::Mat res, int &SAD);` из приложения с исходным кодом программы.

Чтение исходного изображения в матрицу типа `cv::Mat src` происходит с помощью функции `imread` из библиотеки OpenCV
Также используются и другие функции библиотеки, такие как:

`cv::arrowedLine(*)`- Позволяет рисовать стрелки между двумя точками внутри матрицы Mat
`cv::namedWindow("result", CV_WINDOW_AUTOSIZE);`
`cv::imshow("result", cpy);` - Эта комбинация выводит изображение на экран с помощью стандартных средств Windows

Алгоритм работы

Исходное изображение разбивается на блоки размера BLOCK_SIZE, от которых зависит размер зоны поиска AREA_SIZE. После чего осуществляется перебор и сравнение исходного блока с каждым идентичным по размеру блоком из матрицы зоны поиска со сдвигом в один пиксель. Сместим исходное изображение внутри себя, оставив позади белый фон:

Рисунок 1 Исходное изображение

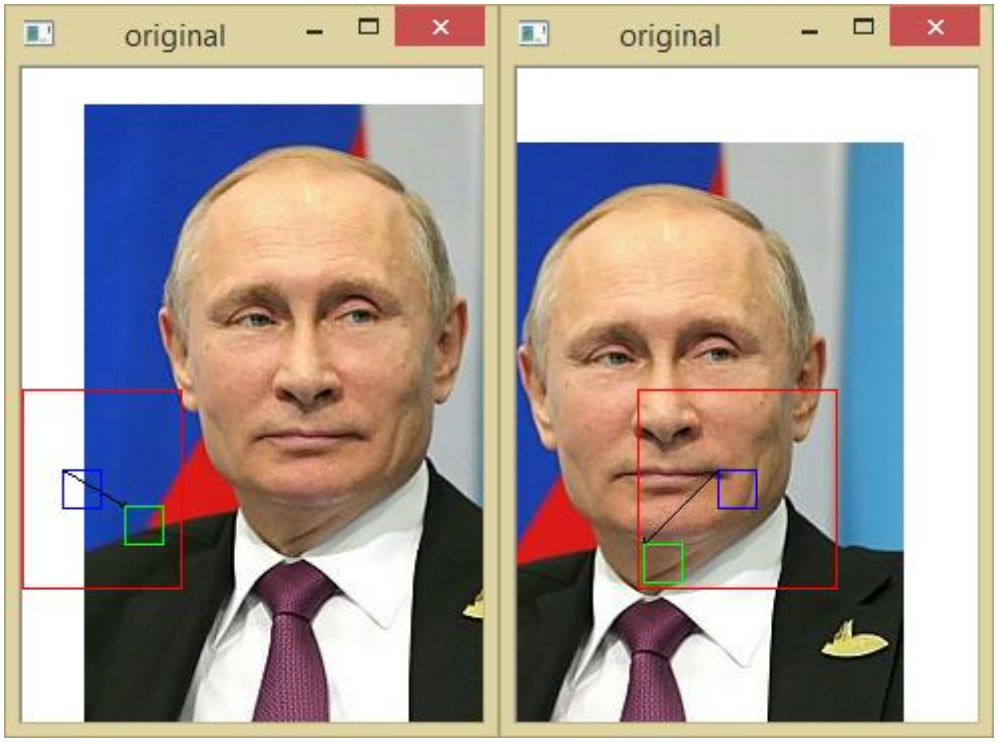

Рисунок 2 Результат поиска сдвинутого блока

На рисунке 2 красным изображена зона поиска блока, в данном случае ее “радиус” равен BLOCK_SIZE*2. Синий прямоугольник – исходный блок, зеленый – сдвинутый. Исходя из результатов, можно заключить, что поиск блока осуществляется корректно.

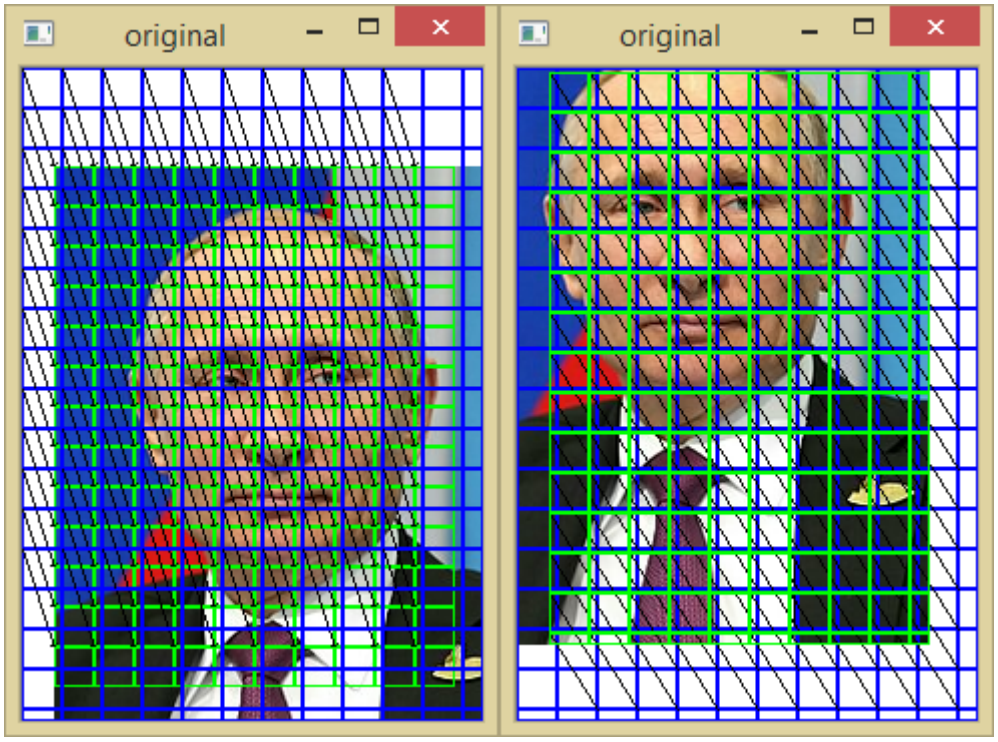

Рисунок 3 Структура изображения и блоков при поиске

Данный рисунок демонстрирует все блоки изображения при поиске смещений. Можно отметить те случаи, когда алгоритм полного перебора на полное соответствие не срабатывает: когда не существует идентичного блока, а именно – на правом изображении слева и на левом изображении справа – цельные блоки из предпоследнего

к краю ряда были урезаны, соответственно, найти их пару не удается. Все целые блоки были корректно определены программой.

Работа с последовательными кадрами

Для данного эксперимента возьмем два кадра из видео:

https://www.youtube.com/watch?v=Jm932Sqwf5E&t=3149s

Рисунок 4 Первый кадр

Рисунок 5 Кадр через 0.25 секунд

В данном видео туча закручивается по часовой стрелке вокруг башни, луч которой играет роль эквалайзера. Дым в нижней части движется к правому краю изображения.

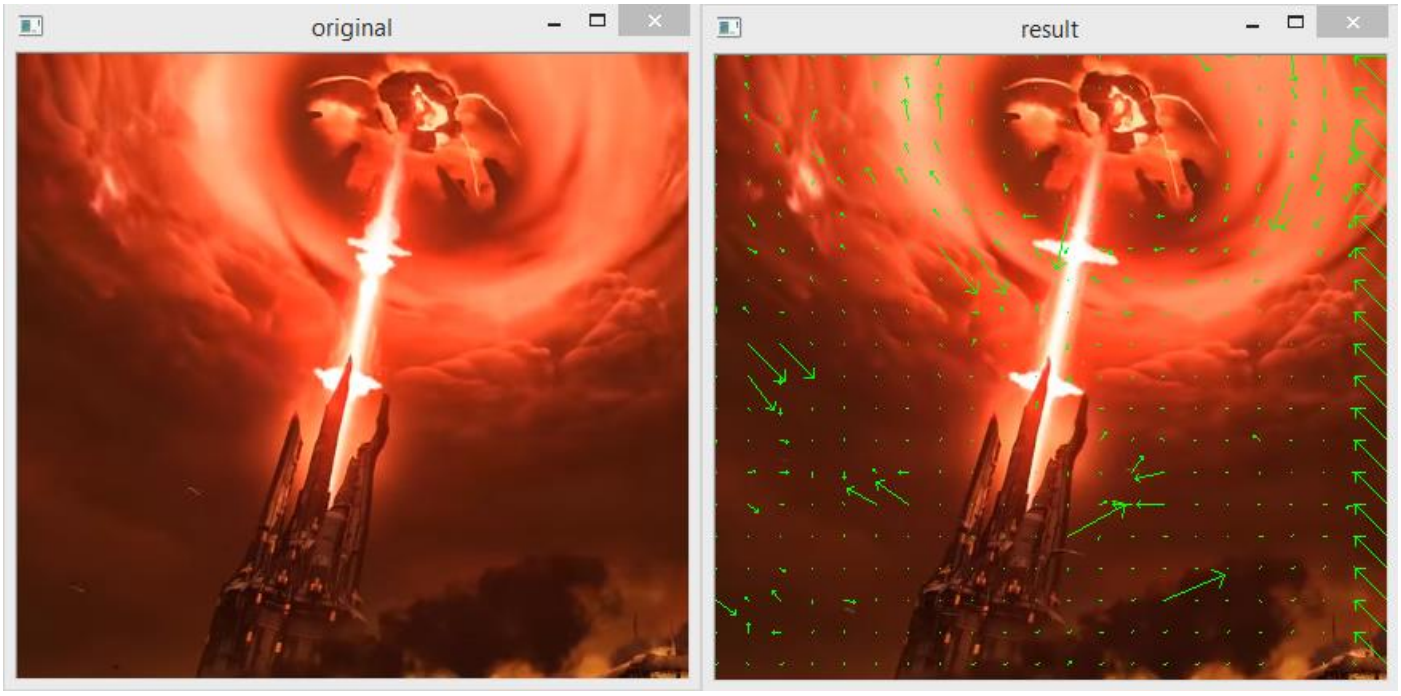

Рисунок 6 Результат работы с последовательными кадрами

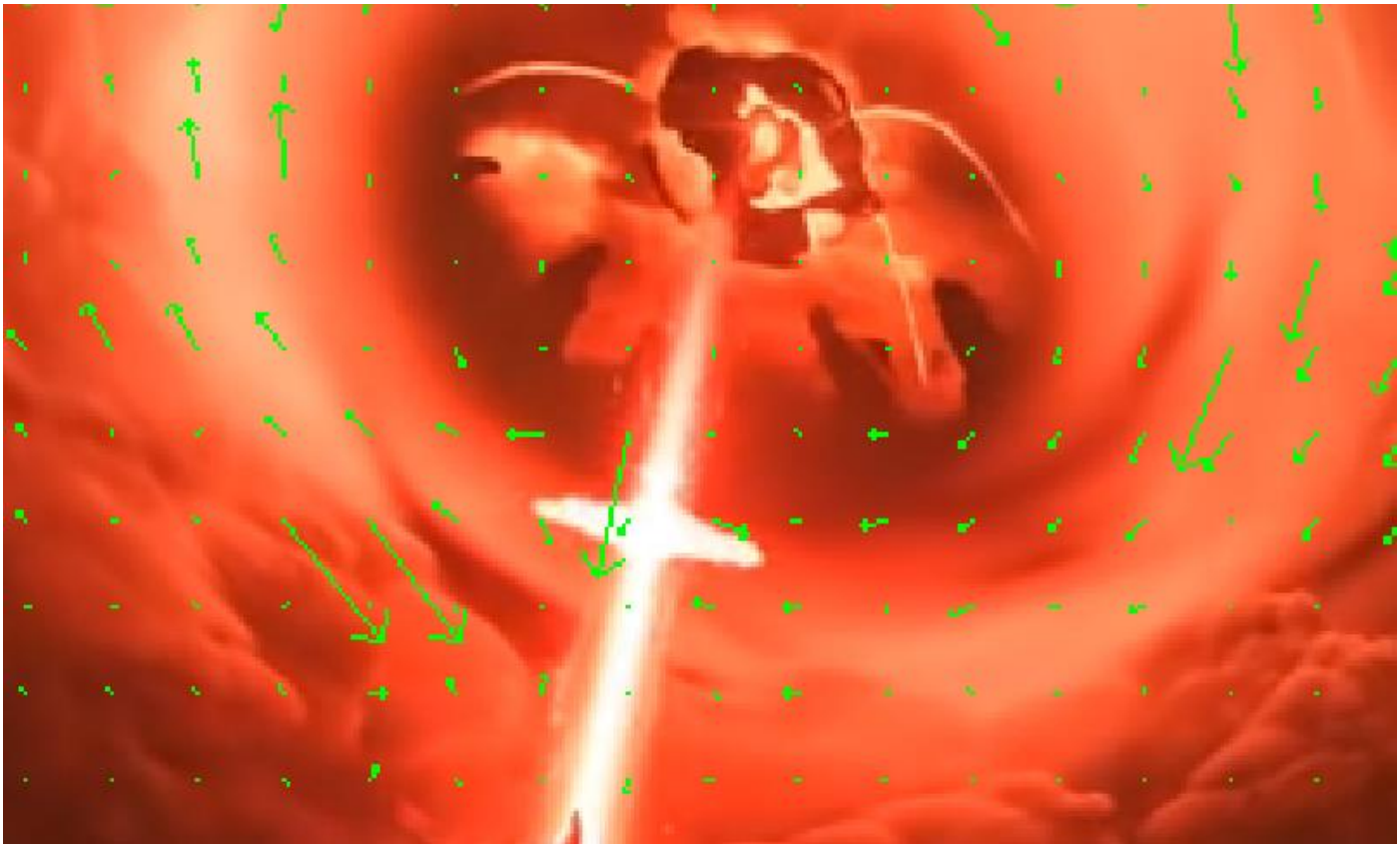

Рисунок 7 Увеличенный вариант результата

Из-за незначительной скорости движения туч, смещение довольно мало. Однако метрика SAD уже позволяет определить направление движений объектов, что видно по направлению стрелок на дыме снизу и самое главное – вокруг тучи, они явно показывают направление ее вращения. При этом статичные объекты, такие как башня и некоторые элементы фона не подвержены изменениям, их блоки отмечены точками – то есть вектор изменения никуда не направлен и совпадает с собственным началом.

Из этого можно сделать вывод, что программа полностью работоспособна.

Выводы

В ходе данной лабораторной работы были изучены основы работы с библиотекой OpenCV, основные типы и методы, которые в ней используются для работы с изображениями.

Также была разработана программа, выполняющая поиск смещений блоков исходного изображения при его изменении. Можно заметить, что результаты работы обладают некоторыми погрешностями, это вызвано принципом работы алгоритма и рядом некоторых параметров.

На ошибку напрямую влияет размер зоны поиска, он подбирается в зависимости от скорости объекта на видео, либо от частоты кадров видео. Если зона будет достаточно большой, можно допустить попадание в нее блока, который проходит метрику SAD лучше, чем исходный перемещенный блок, что вызовет погрешность в его определении

на общем фоне, если при переборе данный блок располагался ниже заданного, так как обход массива зоны поиска осуществляется слева направо.

Крайние блоки, отсеченные от основного изображения и имеющие нестандартный размер (ошибка в у правого края изображения) также могут иметь множество “потомков” с удовлетворительной метрикой SAD ввиду своего размера, что также создает погрешность в однозначном определении перемещения.

Также, значительное изменения внутри блока, а именно возникновение каких-нибудь случайных эффектов, могут изменить представление программы о его положении, при этом объект остается на своем месте.

Еще одним большим минусом данного метода является низкая производительность. На среднем компьютере обработка последовательных полным перебором кадров занимала около 4 минут, что неприемлемо для работы с видеопотоком.

Приложение

Код программы на языке С

```
#include <cstdio>
#include <iostream>
#include <windows.h>

#pragma comment(lib,"I:\\Downloads\\opcv\\opencv\\build\\x64\\vc15\\lib\\opencv_world343d.lib")

#include <opencv2/core.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/imgcodecs.hpp>

#define BLOCK_SIZE 20
#define AREA_SIZE BLOCK_SIZE*1


bool compareSAD(cv::Mat src, cv::Mat res, int &SAD);
int getShiftVectors(char* srcName, char* resName);
void compareImages(cv::Mat &src, cv::Mat &res);
int main(int argc, char *argv[])
{
	if (argc != 3)
	{
		std::cout << "Wrong number of parameters" << std::endl;
		return -1;
	}
	getShiftVectors("frame1.png", "frame2.png");
	return 0;
}


int getShiftVectors(char* srcName, char* resName)
{
	cv::Mat src, res;
	src = cv::imread(srcName, CV_LOAD_IMAGE_COLOR);
	res = cv::imread(resName, CV_LOAD_IMAGE_COLOR);
	compareImages(src, res);
	return 0;
}

void compareImages(cv::Mat &src, cv::Mat &res)
{
	int wSrc = src.rows;
	int hSrc = src.cols;
	size_t sSrc = src.step;
	int wRes = res.rows;
	int hRes = res.cols;
	size_t sRes = res.step;

	cv::Mat cpy;
	res.copyTo(cpy);

	int height = hSrc - hSrc % BLOCK_SIZE;
	int width = wSrc - wSrc % BLOCK_SIZE;
	int h = 0, w = 0;
	int ch, cw;

	for (h = 0; h <= height; h += BLOCK_SIZE)		//Внешние циклы
	{							//
		for (w = 0; w <= width; w += BLOCK_SIZE)	//Задают новый блок
		{
			bool eq = false;
			int bh, bw;
			if ((bh = BLOCK_SIZE) > hSrc - h)
				bh = hSrc - h;
			if ((bw = BLOCK_SIZE) > wSrc - w)
				bw = wSrc - w;

			cv::Rect roi1(h, w, bh, bw);
			cv::Mat srcBlock(src, roi1);

			//cv::rectangle(cpy, roi1, cv::Scalar(255, 0, 0), 1, 8, 0);
			int SADsum = INT16_MAX;
			ch = h;
			cw = w;

			int hal = 0, har = 0; //формирование размера зоны поиска
```

```
                        if ((hal = h - AREA_SIZE) < 0)
                                hal = 0;

                        if ((har = h + AREA_SIZE + bh) >= hSrc)
                                har = hSrc;

                        int wal = 0, war = 0;

                        if ((wal = w - AREA_SIZE) < 0)
                                wal = 0;

                        if ((war = w + AREA_SIZE + bw) >= wSrc)
                                war = wSrc;

                        int i = 0, j = 0;
                        cv::Rect roi3(hal, wal, har - hal, war - wal);
                        //      cv::rectangle(cpy, roi3, cv::Scalar(0, 0, 255), 1, 8, 0);

                        for (i = hal; i < har; i++)             //Данные циклы задают
                        {                                       //
                                for (j = wal; j < war; j++)     //Блок внутри зоны поиска
                                {
                                        if ((i + bh) < hSrc && (j + bw) < wSrc)
                                        {
                                                cv::Rect roi2(i, j, bh, bw);
                                                cv::Mat resBlock(res, roi2);
                                                if (compareSAD(srcBlock, resBlock, SADsum))
                                                {
                                                        ch = i;  cw = j;
                                                }
                                        }
                                }
                        }
                        cv::Rect roi2(ch, cw, bh, bw);
                        cv::Mat resBlock(res, roi2);
                        cv::Point pt1(h, w);
                        cv::Point pt2(ch, cw);
                        cv::arrowedLine(cpy, pt1, pt2, cv::Scalar(0, 255, 0), 1, 8, 0, 0.2);
                        //cv::rectangle(cpy, roi2, cv::Scalar(0, 255, 0), 1, 8, 0);
                }
        }
        cv::namedWindow("result", CV_WINDOW_AUTOSIZE);
        cv::imshow("result", cpy);
        cv::namedWindow("original", CV_WINDOW_AUTOSIZE);
        cv::imshow("original", src);
        cv::waitKey(0);
}

bool compareSAD(cv::Mat src, cv::Mat res, int &SAD)
{
        int SADsum = 0;

        for (int i = 0; i < src.size().height; i++)
        {
                for (int j = 0; j < src.size().width; j++)
                {
                        SADsum += abs(src.at<cv::Vec3b>(i, j)[0] - res.at<cv::Vec3b>(i, j)[0]);
                        SADsum += abs(src.at<cv::Vec3b>(i, j)[1] - res.at<cv::Vec3b>(i, j)[1]);
                        SADsum += abs(src.at<cv::Vec3b>(i, j)[2] - res.at<cv::Vec3b>(i, j)[2]);
                }
        }
        if (SAD > SADsum)
        {
                SAD = SADsum;
                return true;
        }
        return false;
}
```