

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет телекоммуникаций

Кафедра сетей и устройств телекоммуникаций

Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

**АНАЛИЗ АЛГОРИТМОВ СОВМЕЩЕНИЯ ИЗОБРАЖЕНИЙ,
ОСНОВАННЫХ НА ВЫДЕЛЕНИИ ГРАНИЦ**

Студент: гр. 663101 Горбуков Андрей Дмитриевич
Руководитель: Макейчик Екатерина Геннадьевна

Минск 2017

СОДЕРЖАНИЕ

Введение	5
1 Описание среды разработки	7
1.1 Visual Studio	7
1.2 OpenCV	8
2 Теоретический обзор используемых алгоритмов	11
2.1 Оператор Робертса	11
2.2 Оператор Собеля	11
2.3 Оператор Превитта	12
3 Разработка программы	13
3.1 Функциональное проектирование лабораторного стенда	13
4 Анализ полученных результатов	18
4.1 Максимальная амплитуда колебаний камеры	18
4.2 Нахождение части кадра пригодной для обработки	20
4.3 Параметры порогового фильтра	21
Список использованных источников	22
Приложение А (обязательное) Исходный код программного средства	23
Приложение Б (обязательное) Результаты исследования параметров порогового фильтра	27
Приложение В (обязательное) Пример изображения получаемого в результате работы программы	30

ВВЕДЕНИЕ

Гиперспектральная съемка – развивающиеся и перспективное направление дистанционного зондирования земной поверхности [1]. В основе данного вида съемки лежит понятие гиперспектрального изображения. Гиперспектральное изображение – это трехмерный массив данных, который хранит двухмерную информацию о пространственных координатах и дополнительные одномерные данные о спектральных характеристиках точки поверхности. Данная технология широко применяется в различных сферах, таких как:

- сельское хозяйство;
- медицина;
- производство и переработка продуктов питания;
- минералогия;
- физика;
- астрономия;
- химическая визуализация.

Существует множество различных методов получения данного типа изображения, один из которых использует статический Фурье-спектрометр на основе интерферометра Саньяка. Данный метод налагает свои ограничения на данные получаемые во время съемки. Для построения гиперспектрального изображения требуется накопить достаточное количество данных о точке местности, в процессе ее прохождения вдоль направления полета. Ввиду того, что в дальнейшем над полученными сигналами выполняется дискретное преобразование Фурье, к ним предъявляются строгие требования. В частности, необходимо чтобы полученный сигнал относился к одному и тому же участку местности, а его частота дискретизации была равномерной. В лабораторных и других стабильных условиях съемки, когда частоту кадров можно синхронизировать со скоростью полета, достаточно смещать каждый кадр на один пиксель. Таким образом легко получить информацию о яркости любого пикселя под разными углами съемки. При съемке с различных летательных аппаратов вибрации и колебания не позволяют использовать методы применяемые в камерах установленных на спутниках. Для

сведения задачи к вышеописанному случаю требуется отдельно обработать получаемые при съемке местности данные. Для этого необходимо совместить соседние кадры и узнать их относительное перемещение.

Основной задачей данной работы является реализация и анализ различных алгоритмов, позволяющих решить вышеописанную проблему.

Предполагается, что использование детекторов границ позволит выделить особые участки изображений остающиеся на соседних кадрах и отражающие взаимное смещение между ними. Тем самым сокращая перебор.

1 ОПИСАНИЕ СРЕДЫ РАЗРАБОТКИ

В данном разделе будут рассмотрены различные инструменты используемые в ходе выполнения работы, их процесс установки и специфика использования. Также будет произведен краткий сравнительный анализ аналогичных средств, будет дано обоснование выбора именно тех средств, которые использованы в данной работе.

1.1 Visual Studio

Microsoft Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio является одним из лидеров среди интегрированных сред разработки на языке C++. Высокая популярность данной среды выражается в огромном мировом сообществе разработчиков использующих её. Благодаря этому у разработчика, применяющего Visual Studio для решения своих задач, появляется возможность более быстрого решения часто возникающих проблем. Также большинство сторонних библиотек и других инструментов имеют версию специально выпущенную для Visual Studio. Большой ассортимент плагинов и расширений доступен в официальном онлайн магазине Microsoft. Почти все они доступны бесплатно. Данные дополнения позволяют упростить разработку и интегрироваться со сторонними сервисами. В частности в ходе разработки данного проекта применялся плагин VisualSVN, обеспечивающий возможность работать с системой контроля версий SVN.

Visual Studio является коммерческим проектом, однако для некоммерческой разработки предоставляется специализированная версия

Community. По сравнению с версией Professional данная версия обладает ограниченным функционалом, однако в работе этот функционал не использовался. Поэтому разработка велась на версии Visual Studio 2017 Community RC.

Для начала установки требуется скачать установщик с официального сайта Microsoft¹⁾. Там располагается ссылка для скачивания. Скачанный файл запустит процесс установки, предварительно попросив принять условия лицензионного соглашения. Для перехода на следующий этап требуется нажать кнопку «Продолжить». После этого запустится Visual Studio Installer, который служит для установки и модернизации различных версий Visual Studio 2017.

После выбора нужной версии Visual Studio и нажатия кнопки «Установить» откроется окно выбора комплектации среды разработки и дополнительных приложений, показное на рисунке 1.1. Для работы с данным проектом достаточно выбрать пункт «Разработка классических приложений на C++» и предпочтительные языковые пакеты на вкладке «Языковые пакеты». Однако в зависимости от потребностей пользователя можно установить и дополнительные пакеты. Нажатие на кнопку «Установить» запустит процесс установки Visual Studio. После его завершения можно запустить среду разработки. При первом запуске запрашивается вход под аккаунтом Microsoft, если он еще не зарегистрирован то можно, или пропустить данный шаг и работать в неавторизованной версии, или зарегистрироваться на официальном сайте Microsoft.

1.2 OpenCV

OpenCV – выпущенная под лицензией BSD библиотека, а следовательно бесплатная для коммерческого и академического использования. Она имеет интерфейсы для C++, C, Python и Java. Поддерживает такие операционные системы как: Windows, Linux, Mac OS, Android и IOS. OpenCV была разработана для обеспечения эффективных вычислений с ориентацией на приложения реального

¹⁾<https://www.visualstudio.com/ru/downloads/>

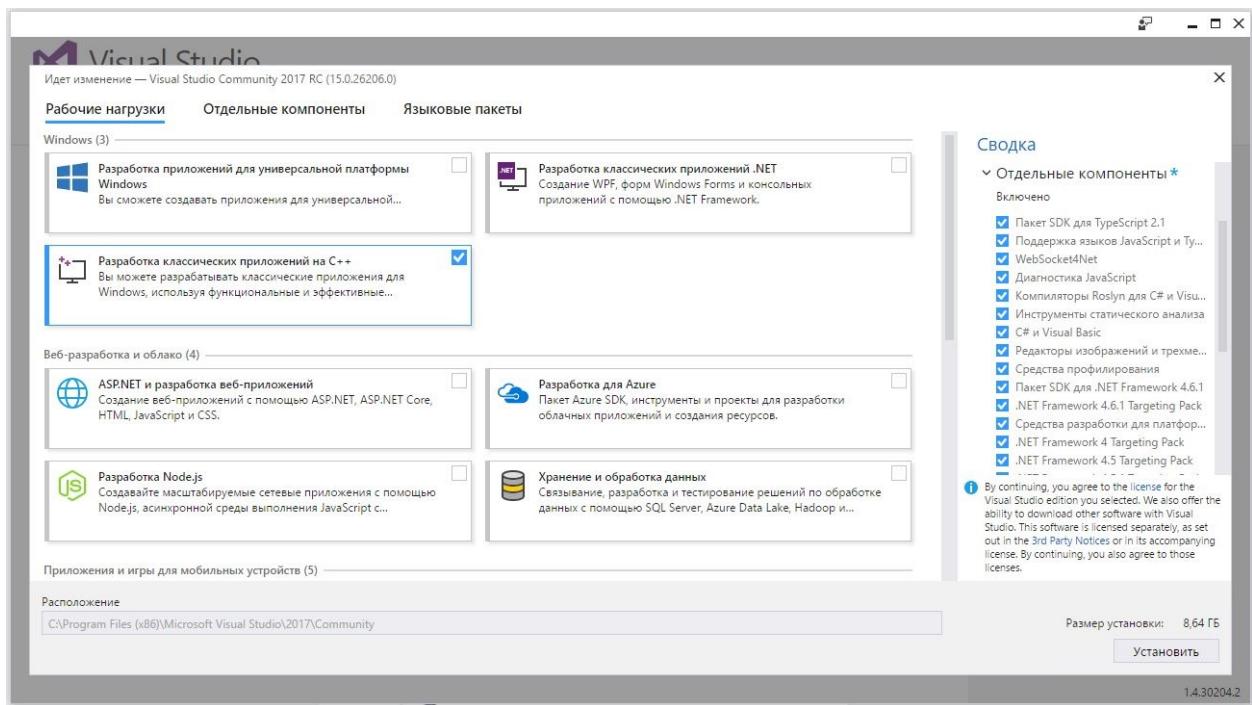


Рисунок 1.1 – Окно выбора комплектации среды разработки и дополнительных приложений

времени. Написанная на оптимизированном С/C++, библиотека способна использовать преимущества многоядерной обработки. Работая вместе с OpenCL она может использовать возможности аппаратного ускорения. Используемая во всем мире, OpenCV насчитывает более 47 тысячи человек в сообществе пользователей и более девяти миллионов скачиваний. Применяется в различных сферах, начиная с интерактивного искусства, обнаружения мин или построения спутниковых карт до современной робототехники [2].

OpenCV – наиболее полная, постоянно развивающаяся библиотека для обработки изображений. Огромное количество функций реализующих различные алгоритмы для обработки и анализа изображений позволяют сосредоточится на решении конкретной задачи, не отвлекаясь на написание заново уже известных классических алгоритмов. Наиболее сильным конкурентом OpenCV в сфере цифровой обработки изображений является MATLAB. Однако он больше подходит для прототипной разработки и исследований алгоритмов из-за удобства отладки, в скорости работы конечного приложения MATLAB уступает возможностям OpenCV. Остальные аналоги реализуют ее функционал лишь частично и

оптимизированы для определенной сферы применения. Исходя из этих преимуществ OpenCV выбрана в качестве основного инструмента для разработки данного проекта.

2 ТЕОРЕТИЧЕСКИЙ ОБЗОР ИСПОЛЬЗУЕМЫХ АЛГОРИТМОВ

В целях анализа возможности применения детекторов границ для решения задачи совмещения изображения были отобраны три оператора. Основанием для отбора этих операторов послужила их простота реализации и скорость работы. Качество найденных границ не учитывалось, ввиду того, что нет необходимости в точности выделенных объектов, требуется только найти особые области сохраняющие свою форму, ориентацию между кадрами и свое положение относительно земной поверхности.

Все ниже описанные операторы используют различные матрицы для приближенного вычисления первой производной изображения, таким образом резкие границы изменения яркости легко обнаружить на основании того факта, что их первые производные превосходят по модулю некоторый порог.

2.1 Оператор Робертса

Один из старейших детекторов разработанный Л. Робертсом в 1963 году [3]. По своей сути вычисляет сумму квадратов разниц между диагонально смежными пикселями. Это может быть выполнено сверткой изображения с двумя ядрами:

$$\begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \quad (2.1)$$

2.2 Оператор Собеля

Дискретный дифференциальный оператор, вычисляющий приближённое значение градиента яркости изображения. Результатом применения оператора Собеля в каждой точке изображения является либо вектор градиента яркости в этой точке, либо его норма [4].

Оператор выполняет свертку двумя ядрами 3×3 для вычисления

приближённых значений производных по горизонтали и по вертикали. Пусть \mathbf{A} – исходное изображение, а \mathbf{G}_x и \mathbf{G}_y приближенные производные вычисляемые следующим образом:

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{A}; \quad \mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{A} \quad (2.2)$$

Где $*$ обозначает двумерную операцию свертки.

Приближенное значение модуля градиента можно вычислить путем взятия квадратного корня из суммы квадратов соответствующих элементов ранее вычисленных матриц.

$$G = \sqrt{{G_x}^2 + {G_y}^2} \quad (2.3)$$

Дополнительно оператор Собеля позволяет вычислить направление градиента:

$$\theta = \arctan \left(\frac{G_y}{G_x} \right) \quad (2.4)$$

2.3 Оператор Превитта

Оператор Превитта использует следующие маски для приближения частных производных:

$$\mathbf{G}_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * \mathbf{A}; \quad \mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{A} \quad (2.5)$$

Данные маски аналогичны оператору Собеля и отличаются только коэффициентом 2 при средних элементах. Данные изменения влияют на уровень сглаживания.

3 РАЗРАБОТКА ПРОГРАММЫ

3.1 Функциональное проектирование лабораторного стенда

Программа реализована с использованием методологий ООП, что позволило повысить коэффициент повторного использования кода и улучшить его читаемость. Разработанные классы, в результате, могут быть использованы для дальнейшей разработки программ схожей тематики. Как можно увидеть из рисунка 3.1 программа состоит из пяти классов, один из которых абстрактный и один статический. В качестве примера в приложении А приведены исходный код основных классов программы.

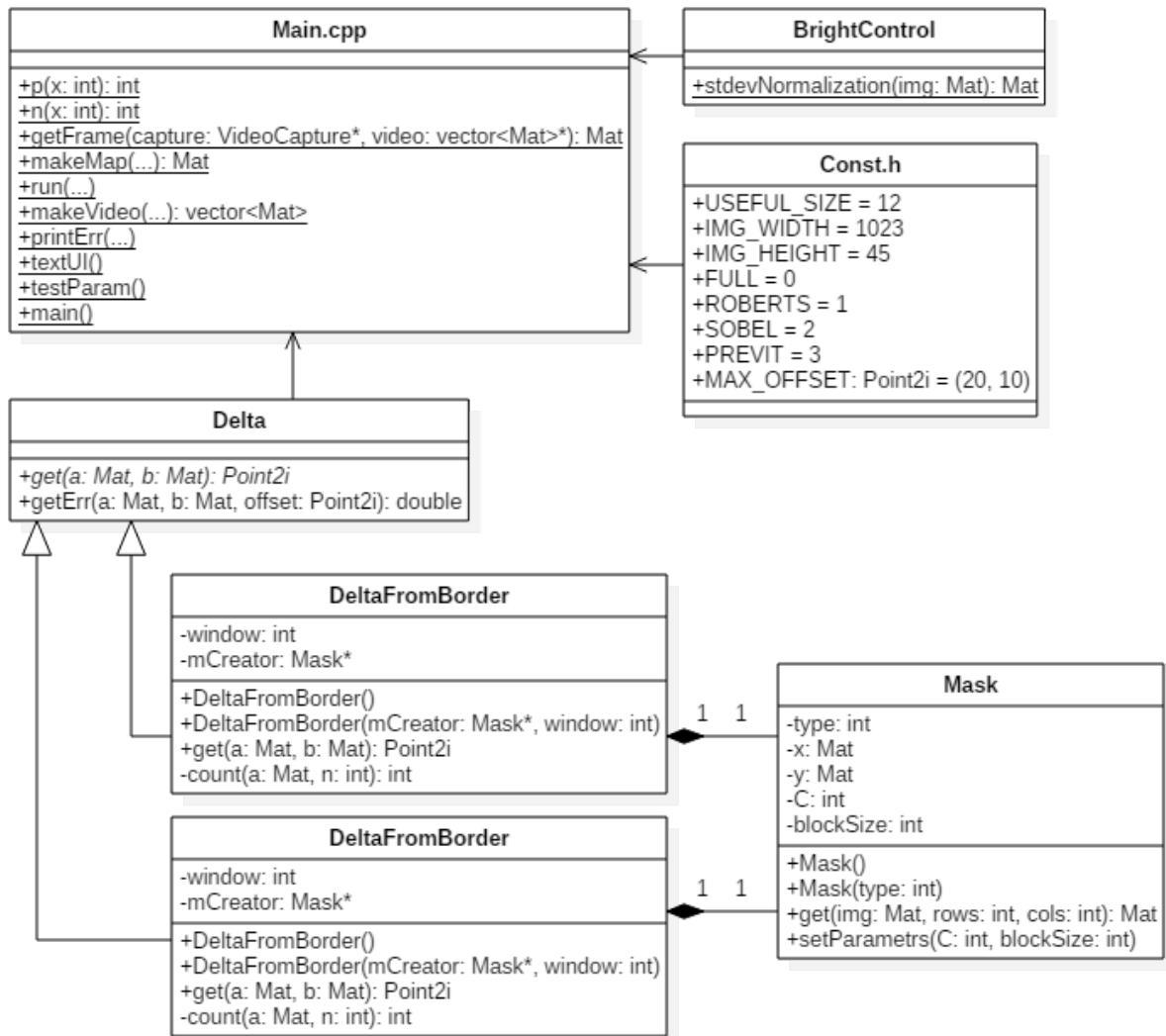


Рисунок 3.1 – Структура программы

3.1.1 Main.cpp – содержит функции, реализующие основную логику программы и чтение/запись файлов. В некоторых из данных функций встречаются одинаковые параметры:

`capture` – входной видеопоток из которого происходит чтение данных.

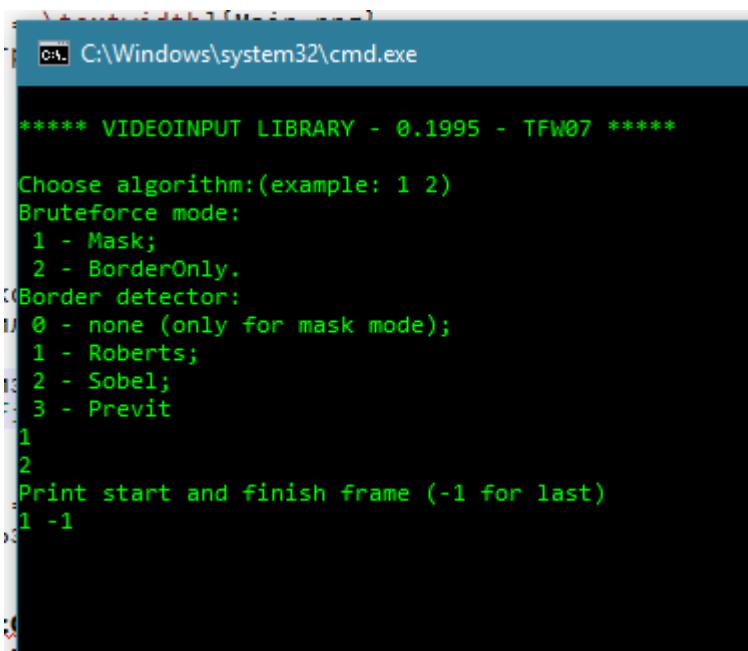
`frameSize` – размер кадра оригинального видео.

`video` – массив изображений в который покадрово записывается видео готовое к обработке.

`offsets` – массив относительных смещений кадров, вычисляемых в процессе работы алгоритмов.

`main()` – точка входа в программу. В зависимости от варианта сборки запускает функцию `textUI()` или `testParam()`.

`textUI()` – реализует текстовый интерфейс для взаимодействия с пользователем. Как видно из рисунка 3.2 пользователь может выбрать тип маски и алгоритм, а также диапазон обрабатываемых кадров.



```
***** VIDEOINPUT LIBRARY - 0.1995 - TFW07 *****  
Choose algorithm:(example: 1 2)  
Bruteforce mode:  
 1 - Mask;  
 2 - BorderOnly.  
Border detector:  
 0 - none (only for mask mode);  
 1 - Roberts;  
 2 - Sobel;  
 3 - Previt  
 1  
 2  
Print start and finish frame (-1 for last)  
 1 -1
```

Рисунок 3.2 – Текстовый пользовательский интерфейс

`testParam()` – служит для многократного запуска алгоритма. Использование данной функции позволяет удобно тестировать алгоритмы при различных параметрах и собирать статистическую информацию.

`vector<Mat> makeVideo(VideoCapture *capture, Size2i *frameSize, int start = 1, int length = -1)` – загружает видео в

виде вектора изображений из видео потока `capture`. Также возвращает размеры кадра. Может принимать номер первого кадра и длину получаемого видео.

`Mat getFrame(VideoCapture *capture, vector<Mat> *video)` – читает следующий кадр из видеопотока, подготовливает его и записывает в массив кадров.

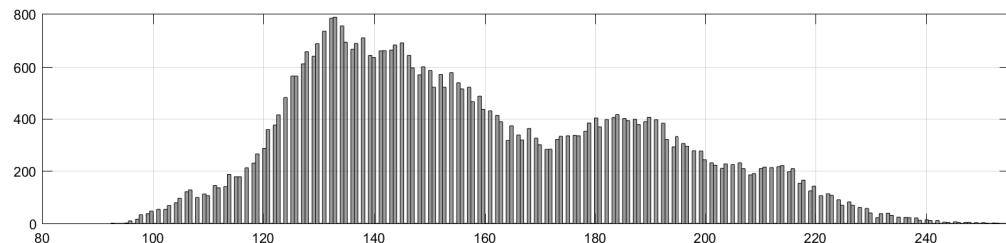
`run(Size2i frameSize, vector<Mat> *video, Mat *map, Delta *delta, vector<Point2i> *offsets)` – обрабатывает видео и запускает создание результирующего изображения `map`. Для этого принимает объект `delta`, определяющий каким алгоритмом будет вычислено относительное смещение кадров.

`Mat makeMap(Size2i frameSize, vector<Mat> *video, vector<Point2i> *offsets, int maxL, int maxR, int maxD, int maxU)` – создает результирующее изображение, для этого требуется узнать максимальные абсолютные смещения кадров во всех четырех направлениях.

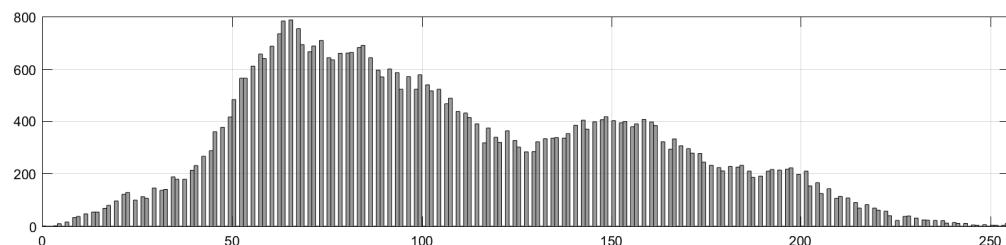
`int printErr(vector<Mat> *video, vector<Point2i> *offsets, ofstream *sout, Mask *mCreator2)` – выводит в текстовый файл дополнительные статистические данные.

3.1.2 Класс BrightControl – содержит единственный статический метод `Mat stdevNormalization(Mat img)` нормализующий изображение так, чтобы яркость любого пикселя подчинялась правилу трех сигм (3σ). Таким образом удается исправить резкие перепады яркости, мешающие работе основных алгоритмов. Примеры исправленных кадров можно увидеть на рисунке ???. Здесь продемонстрированы два наиболее типичных кадра требующих нормализации яркости и их гистограммы до и после обработки.

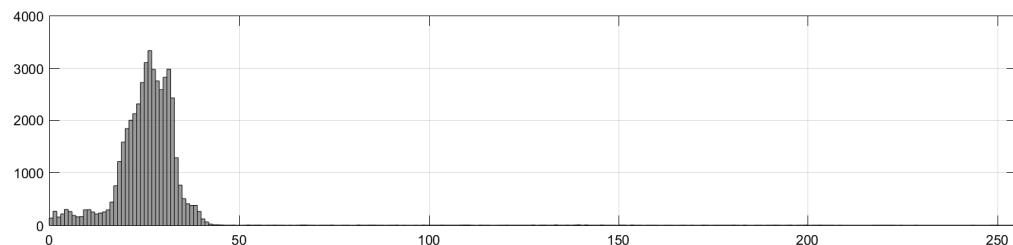
3.1.3 Абстрактный класс Delta – определяет интерфейс для работы с различными алгоритмами, которые вычисляют относительное смещение между соседними кадрами. Использование данного класса позволяет абстрагироваться от конкретной реализации, тем самым дает возможность использовать любой алгоритм без существенных изменений основного кода



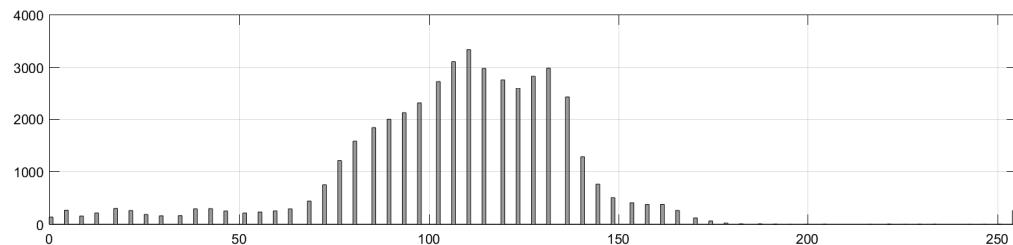
a)



б)



в)



г)

а – снимок и его гистограмма при общем нарушении светочувствительности камеры; б – исправленная версия снимка а и его гистограмма; в – снимок и его гистограмма при засвете камеры отдельным объектом; г – исправленная версия снимка в и его гистограмма

Рисунок 3.3 – Сравнение кадра и его гистограммы до и после нормализации яркости

программы.

Также включает дополнительную функцию `double getErr(Mat a, Mat b, Point2i offset)`. Она служит для вычисления ошибки совмещения изображений.

3.1.4 Класс DeltaBruteForce – реализует интерфейс `Delta`. Перебирает все возможные способы относительного смещения кадров, выбирая то, при котором средняя разница между соответствующими пикселями на кадрах будет минимальной. Учитывает только те пиксели, которые принадлежат области границ, для этого использует обнаруженные границы как маску.

3.1.5 Класс DeltaFromBorder – реализует интерфейс `Delta`, аналогично `DeltaBruteForce` и перебирает все возможные способы относительного смещения кадров. Однако, в этом случае выбирается то при котором область пересечения границ соответствующих кадров максимальна.

3.1.6 Класс Mask – выделяет границы на заданном изображении различными методами, возможно выбирать требуемый метод и параметры пороговой обработки.

3.1.7 Const.h – заголовочный файл содержащий некоторые константы упрощающие разработку и читаемость кода. Определяет константу `DF` позволяющую удобно создавать отладочную версию программы.

4 АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

В ходе разработки и тестирования возникла необходимость в исследовании и анализе некоторых параметров.

4.1 Максимальная амплитуда колебаний камеры

На начальных этапах разработки потребовалось грубо оценить пределы колебаний камеры во время съемки, чтобы существенно сократить количество перебираемых вариантов в вышеописанных алгоритмах перебора.

Для этого было решено использовать метод щелевой съемки. Он позволяет визуально оценить скорость движения камеры по двум осям.

В отличии от обычной фотографии, которая запечатлевает определенный момент времени, щелевая позволяет наблюдать определенное узкое место (щель) на протяжении длительного момента времени. Принцип формирования такого снимка изображен на рисунке 4.1.

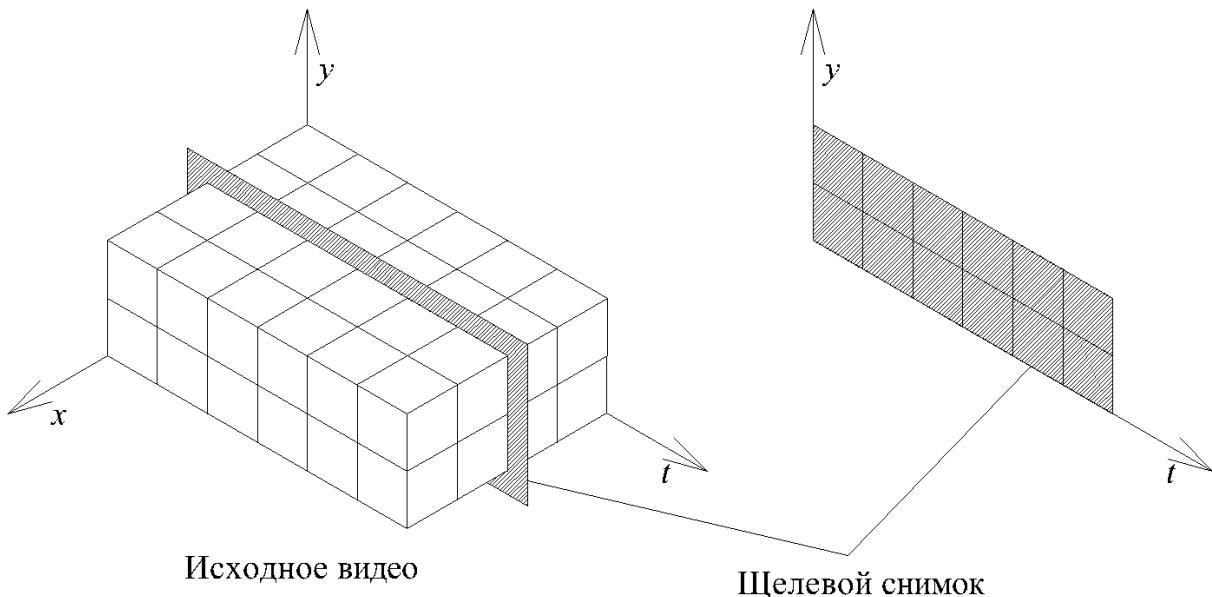


Рисунок 4.1 – Принцип формирования щелевого снимка

В ходе исследования были созданы щелевые снимки для каждой из пространственной оси кадра, изображенные на рисунках 4.2 и 4.3. Здесь красным и зеленым цветом выделены наиболее яркие и темные пиксели каждого кадра, для большего удобства визуального анализа.

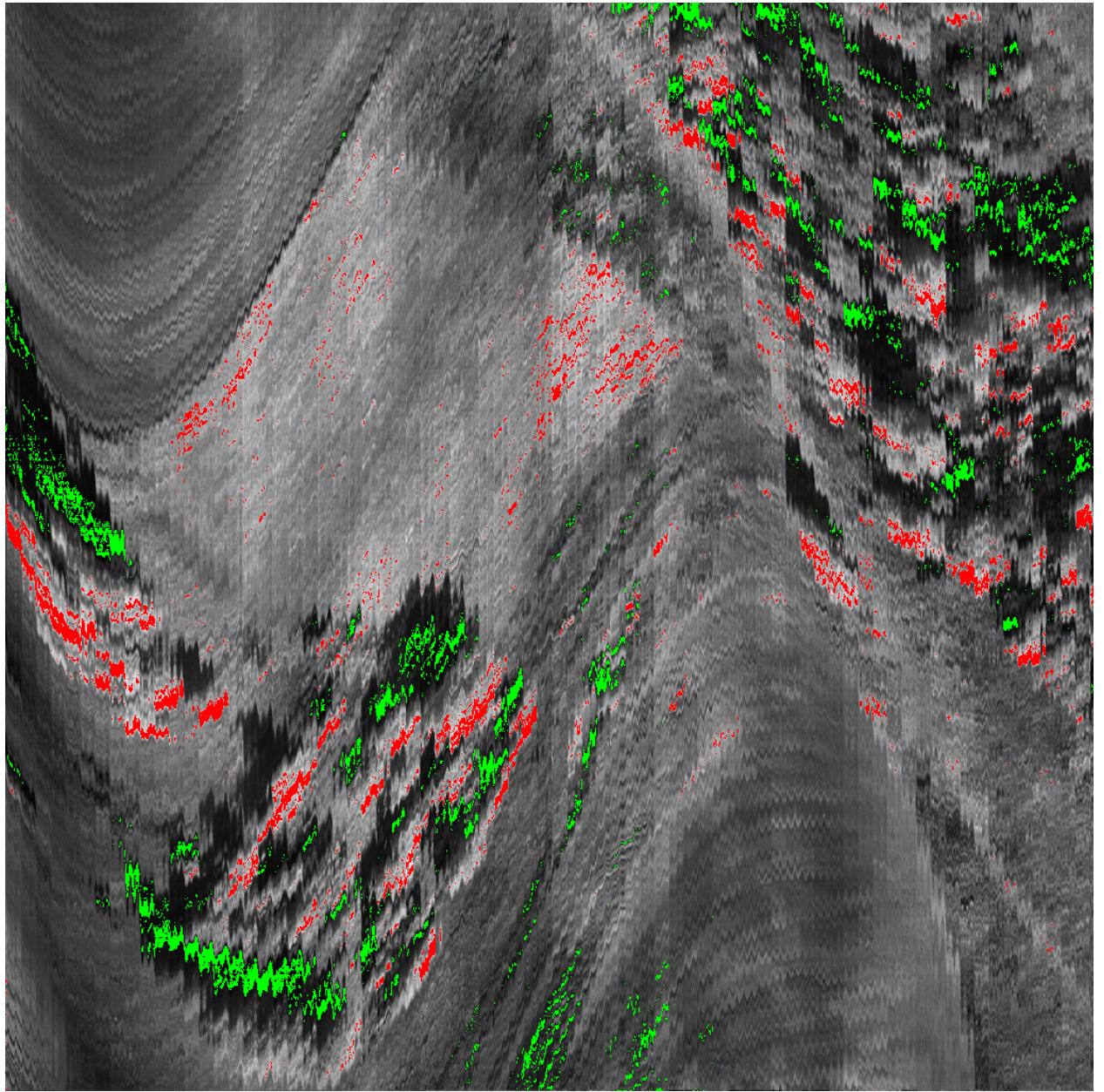


Рисунок 4.2 – Часть полученного щелевого снимка для оси x

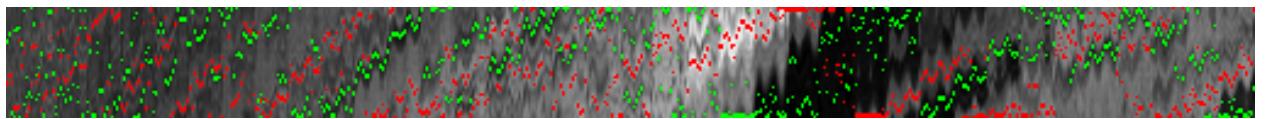


Рисунок 4.3 – Часть полученного щелевого снимка для оси y

По итогом исследования было установлено, что по оси x относительное смещение кадров не превышает 20 пикселей, а по оси y – 10. Полученные ограничения были занесены в файл Const.h.

4.2 Нахождение части кадра пригодной для обработки

Как видно из рисунка 4.4 кадры получаемые со спектрометра имеют свою специфику, в частности можно выделить:

- особый размер кадра, сильно вытянутый в ширину;
- интерференционные полосы в центре кадра.

Для преодоления данных проблем потребовалось дополнительное исследование.

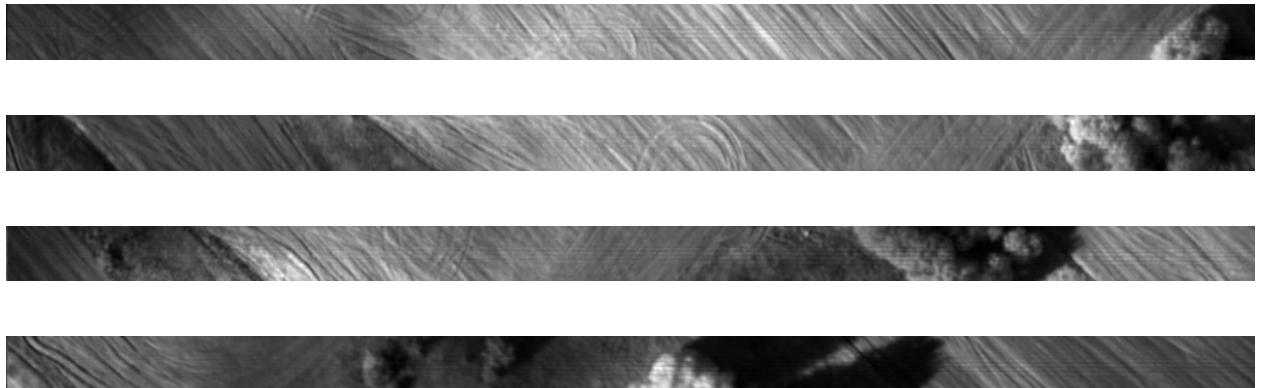


Рисунок 4.4 – Примеры кадров

Широко известно, что оператор Робертса слабо чувствителен к вертикальным и горизонтальным границам. Поэтому была выдвинута гипотеза, что использование данного оператора позволит избежать влияния интерференционных полос. Однако, в ходе исследования, данная гипотеза не подтвердилась, что можно увидеть из рисунка 4.5. Так как избежать влияния интерференции не удалось, то было принято решение выбрать часть кадра не подверженного данным помехам. В ходе визуального анализа было установлено, что верхняя или нижняя полоска кадра шириной в 12 пикселей свободна от сильных периодических шумов и достаточна для совмещения кадров между собой.



Рисунок 4.5 – Пример кадра целиком обработанного оператором Робертса

4.3 Параметры порогового фильтра

На начальных этапах рассматривался вариант использования статического порогового фильтра, однако после серии экспериментов было установлено, что данный вид фильтра не позволяет сформировать маску на разных участках съемки, при этом сохраняя соотношение пикселей маски и всего кадра. Поэтому в дальнейшем исследовании использовался только адаптивный фильтр.

В пороговом фильтре существуют две свободные переменные, влияющие на качество результата: константа C и размер окна. Для исследования влияния данных параметров на результат используются функции `testParam()` и `printErr()`, в связке с дополнительной программой написанной на языке MATLAB. В приложении Б изображены графики показывающие зависимость средней ошибки совмещения кадров от вышеуказанных параметров порогового фильтра для различных алгоритмов выделения границ, а также аналогичный график для средней плотности пикселей. Данная плотность вычисляется как частное от деления количества пикселей принадлежащих найденным границам на общее количество пикселей текущего кадра.

В ходе исследования было установлено, что в среднем обе разновидности алгоритма перебора дают одинаковые результаты. Поэтому предпочтительно использовать алгоритм основанный на поиске наилучшего наложения границ. Среди операторов для поиска границ лучшие результаты показал оператор Собеля с параметрами: $C = -10$ и размером блока равным 7. Примеры изображения получаемого в результате работы данного алгоритма приведены в приложении В.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Цветков, В.Ю. Разработка алгоритмов и программных средств формирования гиперспектральных изображений земли фурье-видеоспектрометра / В.Ю. Цветков, В.К. Конопелько. — Минск : БГУИР, 2016. — 106 с.
- 2 Itseez. OpenCV [Электронный ресурс]. — Режим доступа : <http://opencv.org/>.
- 3 Roberts, Lawrence G. Machine perception of three-dimensional solids: phdthesis. — 1963. — 82 P.
- 4 Sobel, Irwin. History and Definition of the so-called "Sobel Operator more appropriately named the Sobel-Feldman Operator [Электронный ресурс]. — Режим доступа : <https://www.researchgate.net/publication/>.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программного средства

Листинг А.1 – Исходный код класса DeltaBruteForce

```
1 #include "DeltaBruteForce.h"
2
3 #include <opencv2\highgui\highgui.hpp>
4 #include "opencv2/imgproc/imgproc.hpp"
5
6 #include <iostream>
7 DeltaBruteForce::DeltaBruteForce()
8 {
9 }
10
11 DeltaBruteForce::DeltaBruteForce(Mask *mCreator, int window)
12 {
13     this->mCreator = mCreator;
14     this->window = window;
15 }
16
17 DeltaBruteForce::~DeltaBruteForce()
18 {
19 }
20
21 Point2i DeltaBruteForce::get(Mat a, Mat b)
22 {
23     Point2i delta(0, 0);
24     if (b.empty())
25     {
26         return delta;
27     }
28     Mat mask, c;
29     Rect2i roiA, roiB;
30     mask = mCreator->get(a, a.rows, a.cols);
31     if (DF)
32     {
33         Mat buf;
34         resize(mask, buf, Size(0, 0), 2, 2);
35         imshow("w", buf * 255);
36         waitKey(1);
37     }
38     double minRate = 2, rate;
39     for (int i = -MAX_OFFSET.x; i < MAX_OFFSET.x; i++)
40     {
```

```

41     for (int j = -MAX_OFFSET.y; j < MAX_OFFSET.y; j++)
42     {
43         roiA.x = p(i);
44         roiA.y = p(j);
45         roiA.width = a.cols + n(i) - roiA.x;
46         roiA.height = a.rows + n(j) - roiA.y;
47
48         roiB.x = -n(i);
49         roiB.y = -n(j);
50         roiB.width = b.cols - p(i) - roiB.x;
51         roiB.height = b.rows - p(j) - roiB.y;
52
53         absdiff(a(roiA), b(roiB), c);
54
55         rate = mean(c, mask(roiA))[0];
56         if (rate < minRate)
57         {
58             minRate = rate;
59             delta.x = i;
60             delta.y = j;
61         }
62     }
63 }
64 return delta;
65 }
```

Листинг А.2 – Исходный код класса DeltaFromBorder

```

1 #include "DeltaFromBorder.h"
2
3 DeltaFromBorder::DeltaFromBorder()
4 {
5 }
6
7 DeltaFromBorder::DeltaFromBorder(Mask *mCreator, int window)
8 {
9     this->mCreator = mCreator;
10    this->window = window;
11 }
12
13 DeltaFromBorder::~DeltaFromBorder()
14 {
15 }
16
17 Point2i DeltaFromBorder::get(Mat a, Mat b)
18 {
19     Point2i delta(0, 0);
20     if (b.empty())
```

```

21  {
22      return delta;
23  }
24  Mat mask, c, bestC;
25  Rect2i roiA, roiB;
26  a = mCreator->get(a, a.rows, a.cols);
27  b = mCreator->get(b, b.rows, b.cols);
28
29  double maxRate = 0, rate;
30  for (int i = -MAX_OFFSET.x; i < MAX_OFFSET.x; i++)
31  {
32      for (int j = -MAX_OFFSET.y; j < MAX_OFFSET.y; j++)
33      {
34          roiA.x = p(i);
35          roiA.y = p(j);
36          roiA.width = a.cols + n(i) - roiA.x;
37          roiA.height = a.rows + n(j) - roiA.y;
38
39          roiB.x = -n(i);
40          roiB.y = -n(j);
41          roiB.width = b.cols - p(i) - roiB.x;
42          roiB.height = b.rows - p(j) - roiB.y;
43
44          c = (a(roiA) + b(roiB));
45          rate = (double)count(c, 2) / (double)(c.cols * c.rows);
46          if (rate > maxRate)
47          {
48              maxRate = rate;
49              delta.x = i;
50              delta.y = j;
51              bestC = c;
52          }
53      }
54  }
55  if (DF)
56  {
57      Mat buf;
58      resize(bestC, buf, Size(0, 0), 2, 2);
59      imshow("w", buf * 127);
60      waitKey(1);
61  }
62  return delta;
63 }
64
65 int DeltaFromBorder::count(Mat a, int n)
66 {
67     int k = 0;

```

```
68     int nRows = a.rows;
69     int nCols = a.cols;
70     uchar* p = a.data;
71
72     for (unsigned int i = 0; i < nCols*nRows; ++i)
73     {
74         if (*p++ == n) k++;
75     }
76     return k;
77 }
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Результаты исследования параметров порогового фильтра

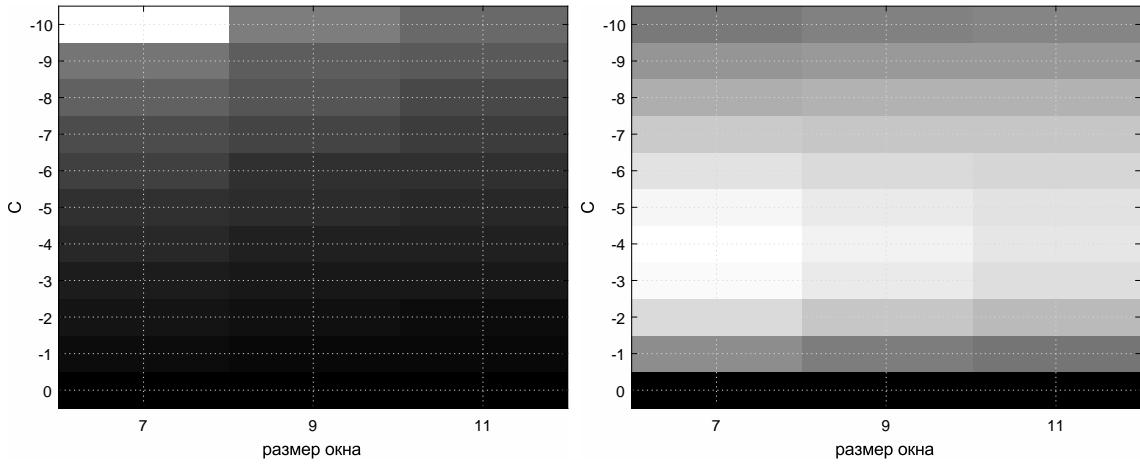


Рисунок Б.1 – Графики зависимостей для алгоритма использующего границы найденные оператором Робертса в качестве маски

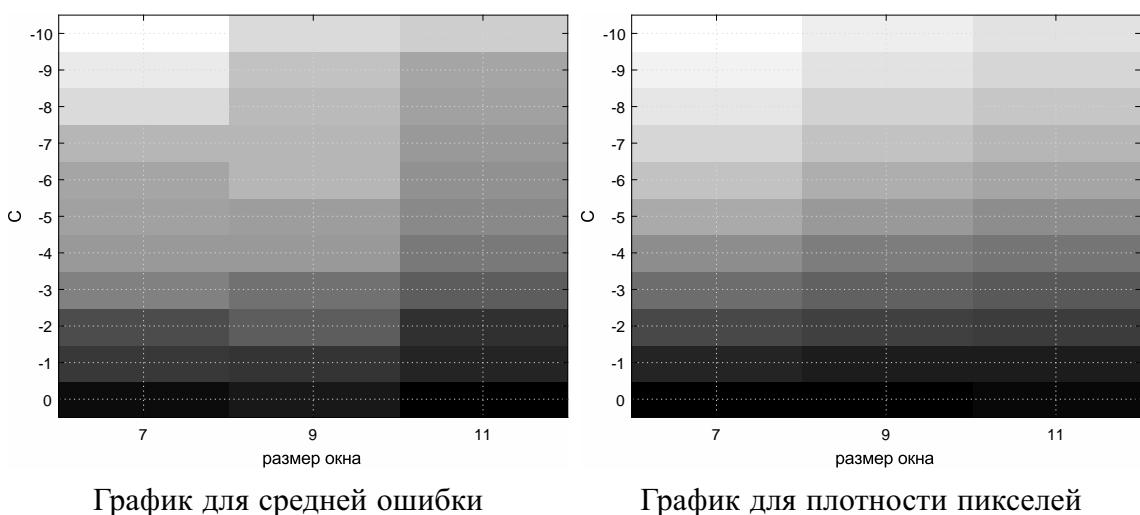


Рисунок Б.2 – Графики зависимостей для алгоритма использующего границы найденные оператором Собеля в качестве маски

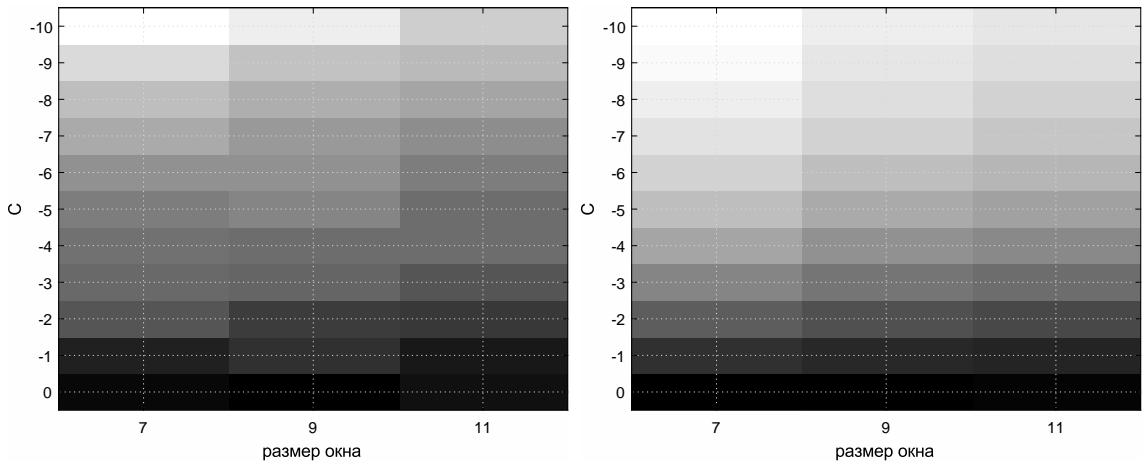


Рисунок Б.3 – Графики зависимостей для алгоритма использующего границы найденные оператором Превитта в качестве маски

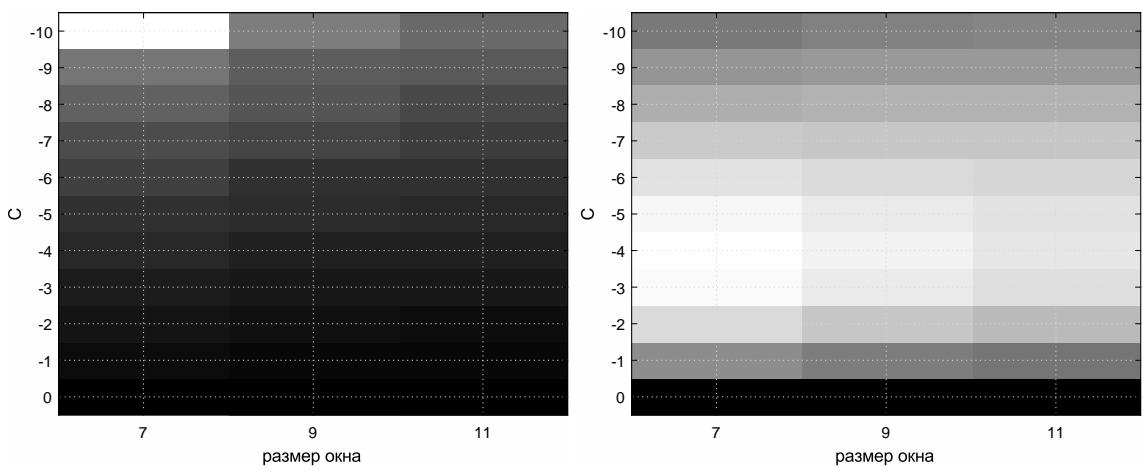


Рисунок Б.4 – Графики зависимостей для алгоритма основанного на поиске наилучшего наложения границ найденных оператором Робертса

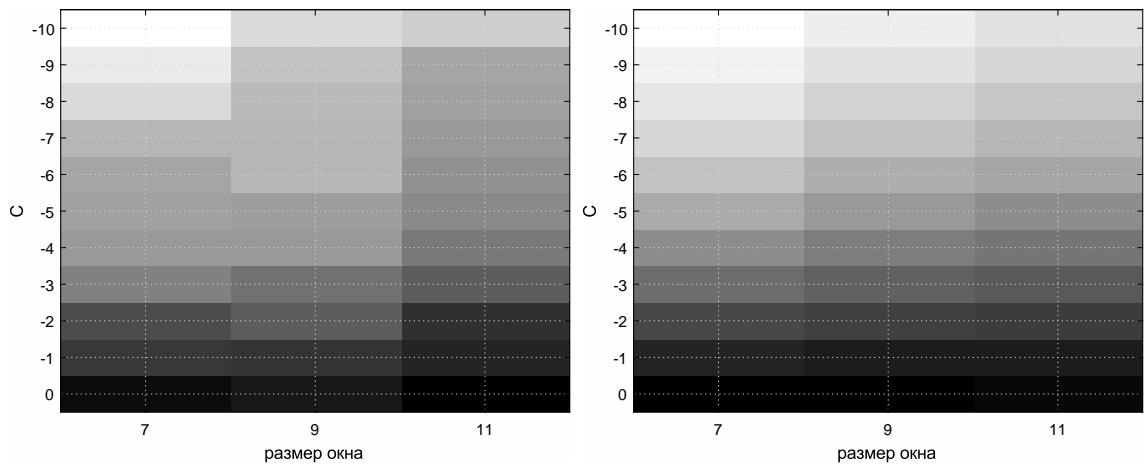


График для средней ошибки

График для плотности пикселей

Рисунок Б.5 – Графики зависимостей для алгоритма основанного на поиске наилучшего наложения границ найденных оператором Собеля

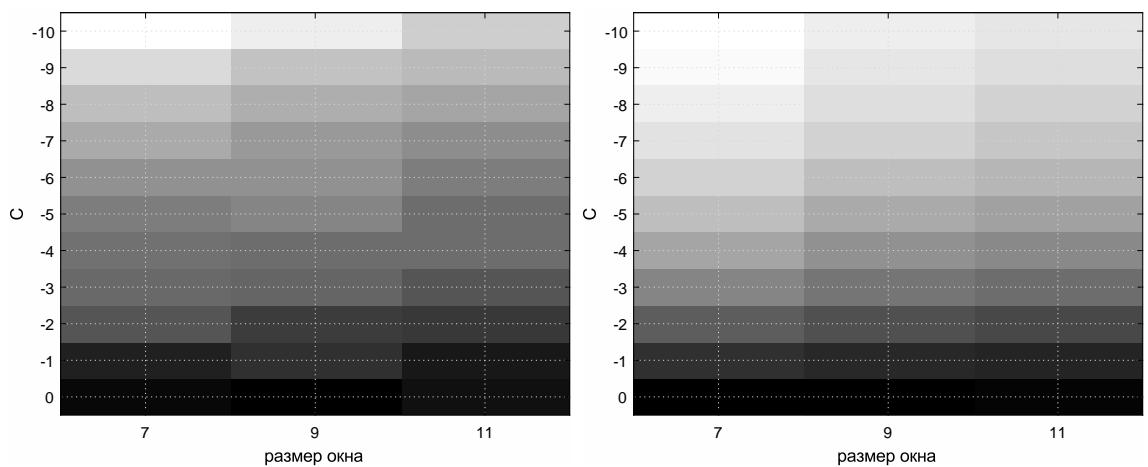


График для средней ошибки

График для плотности пикселей

Рисунок Б.6 – Графики зависимостей для алгоритма основанного на поиске наилучшего наложения границ найденных оператором Преввита

ПРИЛОЖЕНИЕ В

(обязательное)

Пример изображения получаемого в результате работы программы

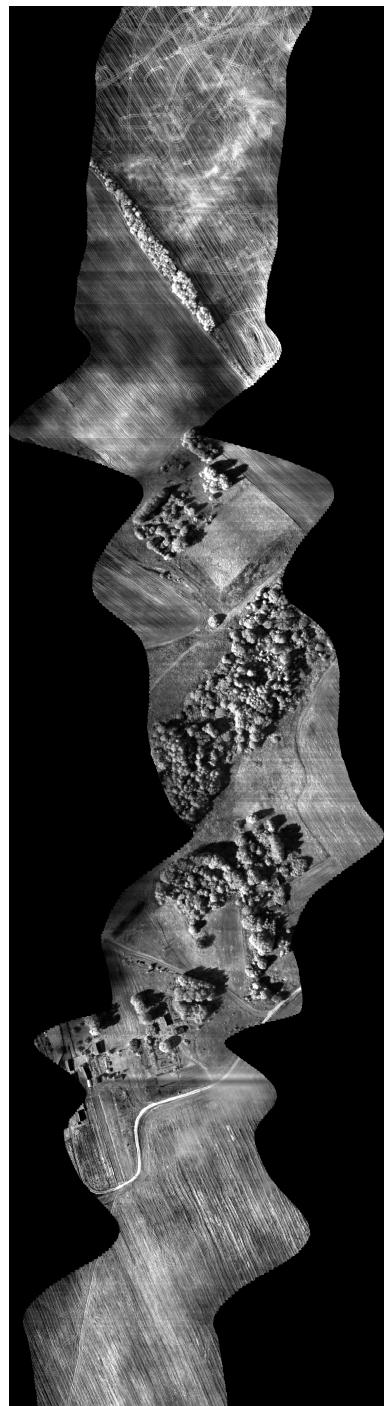


Рисунок В.1 – Пример полученной карты

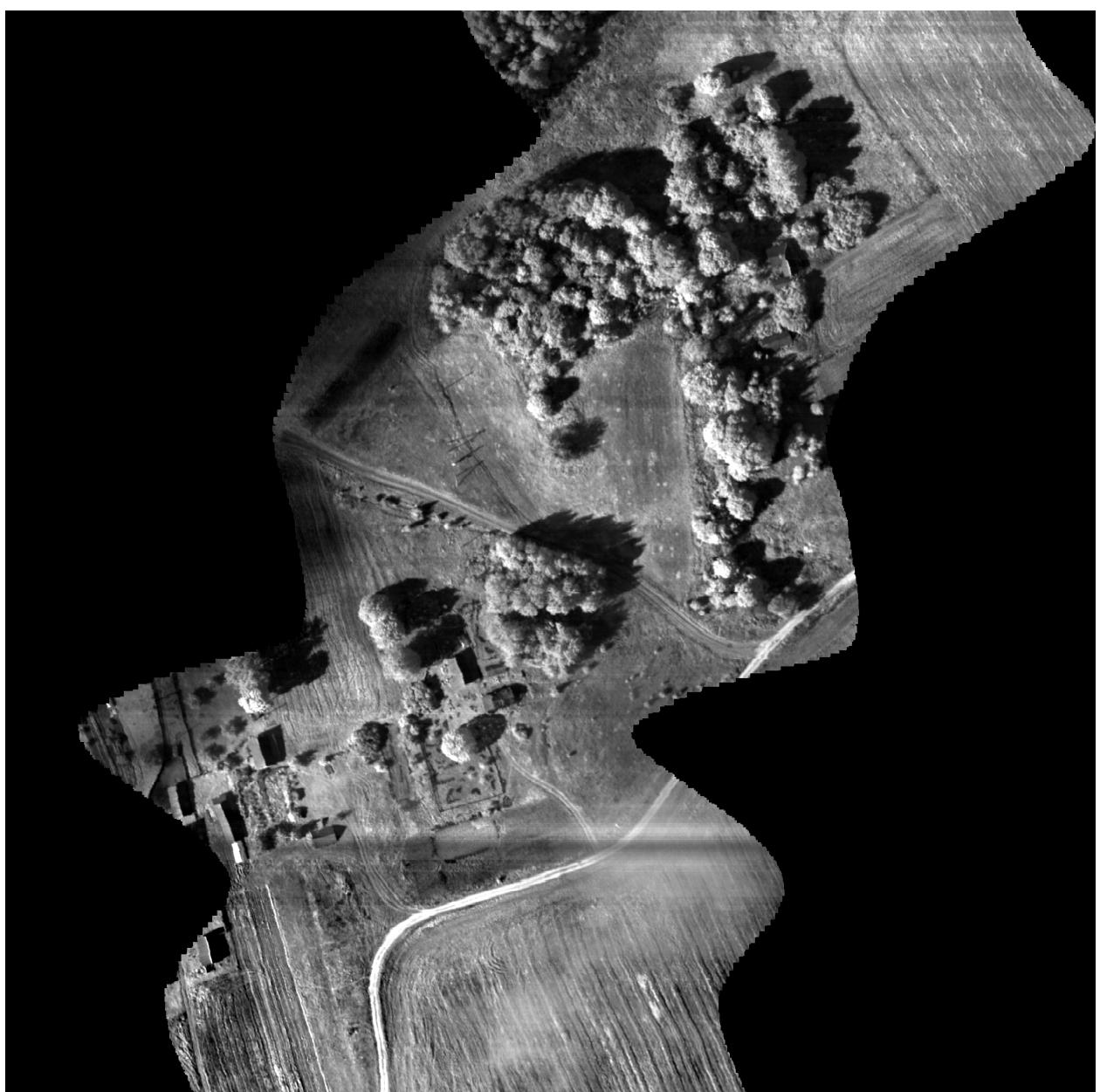


Рисунок В.2 – Часть карты (увеличено)