

**«Санкт-Петербургский государственный электротехнический университет  
«ЛЭТИ» им. В.И.Ульянова (Ленина)»  
(СПбГЭТУ «ЛЭТИ»)**

|                    |   |
|--------------------|---|
| <b>Направление</b> | 11.04.01 – Радиотехника   |
| <b>Программа</b>   | 11.04.01-55 – Инфокоммуникационные<br>технологии анализа и обработки<br>пространственной информации |
| <b>Факультет</b>   | РТ  |
| <b>Кафедра</b>     | ТВ  |

*К защите допустить*

Зав. кафедрой

Лысенко Н. В.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА  
МАГИСТРА**

**Тема: ПРОЦЕДУРА ЦВЕТОВОЙ КАЛИБРОВКИ ДЛЯ  
МУЛЬТИСПЕКТРАЛЬНОЙ СИСТЕМЫ МЕДИЦИНСКОЙ  
ДИАГНОСТИКИ**

|                           |                |              |
|---------------------------|----------------|--------------|
| Студент                   | _____          | Тими́на И.В. |
|                           | <i>подпись</i> |              |
| Руководитель              | _____          | Обухова Н.А. |
| (Уч. степень, уч. звание) | <i>подпись</i> |              |
| Консультант               | _____          | Иванов А.Н.  |
| (Уч. степень, уч. звание) | <i>подпись</i> |              |

Санкт-Петербург

2017

# ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю  
Зав. кафедрой ТВ  
\_\_\_\_\_ Лысенко Н.В.  
« \_\_\_\_ » \_\_\_\_\_ 20\_\_ г.

Студентка      Тимина И.В.      Группа 1104

Тема работы: Процедура цветовой калибровки для мультиспектральной системы  
медицинской диагностики

Место выполнения ВКР: СПбГЭТУ «ЛЭТИ»

Исходные данные (технические требования):

Изображения тестовых таблиц и кольпоскопические изображения, полученные с  
помощью эксперимент. и коммерч. моделей. Предложенная процедура  
калибровки, выполненная на основе МНК с исп. весовой функции должна  
обеспечивать наименьшее значение ошибки цветопередачи на опред. цветах.

Содержание ВКР: Аналитич. обзор методов цветовой коррекции в ПТС,  
разработка процедуры цвет. калибровки для мультиспектр. системы мед.  
диагностики, эксп. исследование предлож. процедуры цвет. калибровки, спец.  
вопросы обесп. безопасности

Перечень отчетных материалов: текст ВКР, иллюстративный материал

Дополнительные разделы: специальные вопросы обеспечения безопасности

Дата выдачи задания  
« \_\_\_\_ » \_\_\_\_\_ 20\_\_ г.

Дата представления ВКР к защите  
« \_\_\_\_ » \_\_\_\_\_ 20\_\_ г.

Студент \_\_\_\_\_

Тимина И.В.

Руководитель \_\_\_\_\_  
(Уч. степень, уч. звание)

Обухова Н.А.

# КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю  
Зав. кафедрой ТВ  
\_\_\_\_\_ Лысенко Н.В.  
« \_\_\_\_ » \_\_\_\_\_ 20\_\_ г.

Студентка Тимина И.В. Группа 1104  
Тема работы: Процедура цветовой калибровки для мультиспектральной  
системы медицинской диагностики

| №<br>п/п | Наименование работ  | Срок<br>выполнения |
|----------|---|--------------------|
| 1        | Обзор литературы по теме работы   | 10.02 –<br>22.03   |
| 2        | Разработка процедуры цветовой калибровки для<br>мультиспектральной системы мед. диагностики | 20.03 –<br>17.04   |
| 3        | Экспериментальное исследование предложенной<br>процедуры цветовой калибровки                | 14.04 –<br>08.05   |
| 4        | Специальные вопросы обеспечения безопасности  | 17.04 –<br>24.05   |
| 5        | Оформление пояснительной записки  | 03.04 –<br>28.05   |
| 6        | Оформление иллюстративного материала  |                    |

Студентка \_\_\_\_\_ Тимина И.В.  
Руководитель \_\_\_\_\_ Обухова Н.А.  
(Уч. степень, уч. звание)

## РЕФЕРАТ

115 с., 35 рис., 2 табл., библи. 25 ист., 1 прил.

### ЦВЕТ, КАЛИБРОВКА, ЦВЕТОКОРРЕКЦИЯ, МАТРИЦА, МЕДИЦИНСКИЕ ИЗОБРАЖЕНИЯ, ТЕСТОВАЯ ТАБЛИЦА, МЕТОД НАИМЕНЬШИХ КВАДРАТОВ

Данная работа посвящена разработке процедуры цветовой калибровки для мультиспектральной системы медицинской диагностики. В работе рассмотрены метод наименьших квадратов и итерационный метод наименьших квадратов, на основе которых может быть реализована цветовая калибровка. Данные методы уменьшают рассогласования цветовых координат, однако, не учитывают степень важности определенных цветов. Поскольку, основой дальнейшей процедуры классификации являются конкретные цвета, то именно для них должны быть обеспечены сопоставимые количественные оценки. Предложенный метод наименьших квадратов с использованием весовой функции позволяет реализовать процедуру калибровки с учётом степени значимости цвета.

## ANNOTATION

115 p., 35 pic., 2 tab., bibl. 25 sources, 1 app.

COLOR, CALIBRATION, COLOR CORRECTION, MATRIX, MEDICAL IMAGES, TEST TABLE, THE METHOD OF LEAST SQUARES

This work is devoted to the development of a color calibration procedure for a multispectral medical diagnostic system. There is a review of the method of least-squares and the iterative method for least-squares based on which a color calibration can be realized. These methods reduce variance of color coordinates but do not take into account the importance of certain colors. Comparable quantitative estimates should have been provided for these certain colors because they are the basis of the further classification procedure. The new calibration procedure can be implemented by using proposed method of the least-squares with the weight function in accordance with degree of significance of the color.

## СОДЕРЖАНИЕ

|   |    |
|---|----|
| ВВЕДЕНИЕ .....  | 8  |
| 1. АНАЛИТИЧЕСКИЙ ОБЗОР МЕТОДОВ<br>ЦВЕТОВОЙ КОРРЕКЦИИ В ПТС.....   | 10 |
| 1.1. Цвет как основа сегментации и классификации в ПТС .....  | 10 |
| 1.2. Особенности классификации и сегментации изображений<br>в ПТС медицинского назначения .....               | 11 |
| 1.3. Цветовая коррекция .....   | 13 |
| 1.3.1. Коррекция на основе спектральных кривых .....  | 13 |
| 1.3.2. Матричная цветокоррекция .....   | 14 |
| 1.4. Применение специальных мишеней для цветокоррекции.....   | 17 |
| 1.4.1. Виды серых шкал .....  | 19 |
| 1.4.2. Виды цветовых шкал .....   | 20 |
| 1.5. Выводы и постановка задачи .....   | 22 |
| 2. РАЗРАБОТКА ПРОЦЕДУРЫ ЦВЕТОВОЙ КАЛИБРОВКИ<br>ДЛЯ МУЛЬТИСПЕКТРАЛЬНОЙ СИСТЕМЫ МЕДИЦИНСКОЙ<br>ДИАГНОСТИКИ..... | 23 |
| 2.1. Цветовая калибровка на основе матрицы преобразования цветовых<br>пространств .....                       | 23 |
| 2.2. Идентификация коэффициентов матрицы цветовой<br>калибровки .....   | 25 |
| 2.3. Процедура цветовой калибровки с учётом степени<br>значимости цвета.....                                  | 28 |
| 2.3.1. Процедура формирования весовых коэффициентов.....  | 28 |
| 2.3.2. Программная реализация процедуры формирования весовых<br>коэффициентов .....                           | 30 |
| 3. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ ПРЕДЛОЖЕННОЙ<br>ПРОЦЕДУРЫ ЦВЕТОВОЙ КАЛИБРОВКИ .....                         | 34 |
| 3.1. Мультиспектральные ПТС .....   | 34 |
| 3.2. Мультиспектральная ТВ система диагностики рака шейки матки ..  | 35 |

|   |    |
|---|----|
| 3.2.1. Общие сведения .....   | 35 |
| 3.2.2. Аппаратная реализация мультиспектрального комплекса<br>LuxCol.....                   | 37 |
| 3.2.3. Обоснование необходимости калибровки.....  | 38 |
| 3.3. Методика эксперимента .....  | 40 |
| 3.3.1. Описание моделирующего программного обеспечения.....                                 | 40 |
| 3.3.2. Критерии оценки качества цветовой калибровки .....                                   | 45 |
| 3.4. Анализ результатов экспериментального исследования .....                               | 49 |
| 4. СПЕЦИАЛЬНЫЕ ВОПРОСЫ ОБЕСПЕЧЕНИЯ<br>БЕЗОПАСНОСТИ .....                                    | 53 |
| 4.1. Эргономический аспект разработки.....  | 53 |
| 4.2 Проектирование на основе точного определения пользователей, задач и<br>среды.....       | 54 |
| 4.3. Условия использования .....  | 55 |
| 4.4. Представления пользователя и разработчика о качестве программного<br>обеспечения ..... | 55 |
| 4.5. Функциональные требования к программному обеспечению .....                             | 56 |
| 4.6. Интерфейс пользователя .....   | 57 |
| 4.7. Принципы организации диалога.....  | 60 |
| 4.7.1. Приемлемость организации диалога для производственного<br>задания.....               | 60 |
| 4.7.2. Информативность .....  | 61 |
| 4.7.3. Соответствие ожиданиям пользователей .....   | 62 |
| 4.7.4. Адаптируемость к индивидуальным особенностям<br>пользователя .....                   | 63 |
| ЗАКЛЮЧЕНИЕ .....  | 66 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....  | 68 |
| ПРИЛОЖЕНИЕ .....  | 70 |

## ВВЕДЕНИЕ

Автоматизированные информационные системы давно прочно вошли в нашу жизнь. С появлением мощных вычислительных машин, спектр выполняемых ими задач существенно расширился. Главная задача автоматизированных информационных систем заключается в автоматизации деятельности, связанной с хранением, передачей и обработкой информации. Одними из наиболее востребованных информационных систем являются компьютерные системы обработки изображений. Сегодня в технике нет почти ни одной области, которую в той или иной мере не затрагивала бы цифровая обработка изображений: микроскопия, астрономия промышленное производство, метеомониторинг [1]. Одной из важнейших областей применения цифровой обработки изображений является медицинская диагностика. В медицине формирование изображений происходит с помощью множества источников: акустические и ультразвуковые колебания, электронные пучки, гамма-излучение, рентгеновские лучи, компьютерное синтезирование изображений. В большинстве случаев, для удобства работы с данными изображениями необходимо провести предварительную обработку. Например, улучшение качества изображений делает его более пригодным для конкретного применения, восстановление поврежденных изображений повышает визуальное качество, а распознавание отдельных элементов позволяет идентифицировать объекты на изображениях.

Идентификация и сегментация объектов на изображении позволяет решать широкий круг задач от распознавания лиц и автомобильных номеров до распознавания целых ситуаций и анализа поведения людей. Как правило, объекты исследования на медицинских изображениях имеют резко неоднородную, сложную структуру и требуют от специалистов большого опыта для выявления тех или иных особенностей, позволяющих поставить определенный диагноз. Использование вычислительной техники и математических методов в этой отрасли позволяет автоматизировать этот процесс, повысить точность результатов исследований и сделать их более объективными [2]. Большинство методов классификации патологий используют цветовой признак как основу для оценки степени поражений и сегментации областей интереса. Однако, в медицинских системах не всегда используются приборы, которые позволяют получить на выходе достоверное изображение. Поскольку, для классификации необходимо достоверная



передача цветов, на основе которых выполняется классификация, то необходимо выполнить процедуры цветокоррекции и цветовой калибровки.

# 1. АНАЛИТИЧЕСКИЙ ОБЗОР МЕТОДОВ ЦВЕТОВОЙ КОРРЕКЦИИ В ПТС

## 1.1. Цвет как основа сегментации и классификации в ПТС

За последние годы область применения цифровой обработки изображений значительно расширилась. Этому способствовало повышение скорости работы, а также уменьшение стоимости и размеров цифровых вычислительных машин и технических средств обработки сигналов. Методы обработки изображений играют значительную роль в научных исследованиях, промышленности, медицине, космических исследованиях и информационных системах. Примерами применения этих методов могут служить цифровая передача изображений с космических кораблей, повышение четкости изображений, создаваемых электронным микроскопом, коррекция искажений изображений, принимаемых из космоса, автоматическое составление карт по аэрофотоснимкам [3]. Сегодня в медицинской технике широко применяются системы формирования изображения, его преобразования в цифровую форму, визуализация и документирование путем введения в компьютер изображений с помощью специализированных устройств захвата видео. Важным предметом исследований и разработок является улучшение качества и автоматизация обработки медицинских изображений, включая изображения, создаваемые электронными микроскопами, рентгеновскими аппаратами, томографами и т.д. При этом под обработкой изображений понимается не только улучшение зрительного восприятия изображений, но и классификация и сегментация объектов [4]. Поэтому, одной из основных задач обработки и анализа изображений является сегментация - разбиение изображения на однородные по некоторым признакам области с одинаковыми свойствами, для которых выполняется определенный критерий однородности, например, выделение на изображении областей приблизительно одинаковой яркости. Количество признаков, которыми могут отличаться друг от друга участки различных типов, очень велико. Обычно в качестве критериев однородности используются яркостный и цветовой признаки, текстурный признак, а также признаки формы или движения. Применение сегментации по тому или иному признаку зависит от решаемой задачи.

Сегменты, обладающие определенными характеристиками объектов исследования, могут быть дополнительно сгруппированы в объекты

определенного типа. В свою очередь, данные объекты могут быть сгруппированы в классы, представляющие собой реальные объекты или процессы. Задачей классификации является нахождение искомого объекта на изображении или видео последовательности и определение принадлежности объекта к тому или иному классу путем анализа вектора значений признаков. Употребляется также термин распознавание образов (pattern recognition), который можно считать синонимом. Под образом понимается описание объекта или процесса, позволяющее выделять его из окружающей среды и группировать с другими объектами или процессами для принятия необходимых решений. В общем случае, в качестве образа может рассматриваться любая информационная модель объекта или процесса абстрактного или реального мира. Информацию о связи между значениями признаков объекта и его принадлежностью к определенному классу алгоритм распознавания должен извлечь из обучающей выборки объектов, для которых известны либо значения признаков и классов, либо только значения их признаков [5].

Для выявления признаков присущим искомым классам применяются различные методы, например:

- расстояние Махаланобиса;
- метод опорных векторов;
- регрессионные модели;
- леса случайных деревьев;
- нейронные сети.

Выбор того или иного метода сегментации зависит от особенностей поставленных задач. Так же важными критериями при выборе являются параметры производительности и качество работы метода.

## 1.2. Особенности классификации и сегментации изображений в ПТС медицинского назначения

В медицине признаки могут характеризовать собой результаты обследований, симптомы заболевания и применявшиеся методы лечения. Накопив достаточное количество прецедентов в электронном виде, можно классифицировать вид заболевания (дифференциальная диагностика), находить синдромы, наиболее характерные для данного заболевания по совокупности симптомов и оценивать риск осложнений. Получением

признаков является набор процедур, в результате действия которых происходит сопоставление изображению набора числовых характеристик - вектора признаков. В качестве признаков наиболее часто используют морфологические и цветовые (яркостные) характеристики изучаемых объектов.

В задаче сегментации медицинских изображений реализуют разделение на области, имеющие близкие цветовые координаты, например, при выделении областей с тканями, отличающимися от основной части изображения. В других случаях цветовая сегментация имеет более частный характер - выделение областей с заранее известным цветом [6]. Например, при использовании метода обнаружения подобия определяется область сегментации, которая делит изображение на регионы, основанные на желаемом свойстве - цвете или интенсивности. Каждый пиксел рассматривается независимо от остальных, затем, используя методы распознавания, определяется, относится ли данный участок к патологии.

В системах медицинского и биомедицинского назначения объекты не имеют выраженной характерной формы и лишены движения, поэтому в прикладных телевизионных системах в качестве основного признака используется цвет. Следовательно, обязательным условием перед применением классификации является калибровка цветовых признаков. Если имеются две камеры, для которых выполнена процедура цветокоррекции, а именно минимизированы ошибки воспроизведения цветов каждой камерой, то это не гарантирует, что цвета, передаваемые первой камерой, будут идентичны с определенным допуском к цветам второй камеры. Необходимо минимизировать ошибки цветопередачи именно между этими двумя камерами - выполнить калибровку второй камеры по первой камере. Только после этого можно использовать изображения, получаемые этими камерами, для задач классификации.

### 1.3. Цветовая коррекция

#### 1.3.1. Коррекция на основе спектральных кривых

В медицинских ПТС обязательно выполняется предварительная обработка изображений, в которую входит устранение артефактов или другие специальные процедуры. Улучшение качества изображений предполагает устранение нежелательных эффектов, которые могут быть вызваны неравномерными источниками света, настройками камер, большими расхождениями в цветах, а также клиническими условиями наблюдения. Растяжка или выравнивание гистограмм обеспечивают максимально-возможный диапазон яркостей изображения, но не могут обеспечить корректное воспроизведение цвета.

При использовании цвета в качестве основного признака при анализе изображений, необходимо наличие сопоставимости его количественных оценок у изображений, полученных разными камерами или полученных в разных условиях наблюдения. Выполнение этого условия определяет необходимость проведения специальной процедуры – цветовой калибровки.

Возможны два способа реализации цветовой калибровки камер:

- сведение спектральных кривых;
- применение матрицы линейного преобразования.

Базовой методикой калибровки камер является сведение их спектральных кривых. В ходе реализации процедуры проводится синтез новых спектральных характеристик для калибруемой камеры «В» максимально близких к спектральным характеристикам опорной камеры «А» путем линейной комбинации исходных спектральных характеристик камеры «В».

Пусть  $r_A(\lambda)$ ,  $g_A(\lambda)$ ,  $b_A(\lambda)$  и  $r_B(\lambda)$ ,  $g_B(\lambda)$ ,  $b_B(\lambda)$  – спектральные характеристики сенсоров А и В, соответственно. Новые спектральные характеристики сенсора В  $r'_B(\lambda)$ ,  $g'_B(\lambda)$ ,  $b'_B(\lambda)$  формируют путем линейной комбинации его исходных спектральных характеристик:

$$\begin{aligned}r'_B(\lambda) &= a_1 r_B(\lambda) + c_1 g_B(\lambda) + d_1 b_B(\lambda), \\g'_B(\lambda) &= a_2 r_B(\lambda) + c_2 g_B(\lambda) + d_2 b_B(\lambda), \\b'_B(\lambda) &= a_3 r_B(\lambda) + c_3 g_B(\lambda) + d_3 b_B(\lambda),\end{aligned}$$

где  $a_i + c_i + d_i = 1$ ,  $i = 1...3$ .

Коэффициенты  $a_i, c_i, d_i$ , где  $i=1...3$  определяют из условия минимума расстояния между новыми синтезированными характеристиками сенсора В  $r'_B(\lambda), g'_B(\lambda), b'_B(\lambda)$  и характеристиками опорного сенсора А  $r_A(\lambda), g_A(\lambda), b_A(\lambda)$ . Если каждую из спектральных характеристик представить как вектор отсчетов

$$\begin{aligned} r_s(\lambda) &= [r_s^N, r_s^N, \dots, r_s^N], \\ g_s(\lambda) &= [g_s^1, g_s^2, \dots, g_s^N], \\ b_s(\lambda) &= [b_s^1, b_s^2, \dots, b_s^N], \end{aligned}$$

где  $s = A, B$ .

то задача определения коэффициентов формализуется следующим образом:

$$\begin{aligned} \min_{a_1, c_1} \sum_i^N & \left( r_A^i - a_1 r_B^i + c_1 g_B^i + (1 - a_1 - c_1) \cdot b_B^i \right)^2 ; \\ \min_{a_2, c_2} \sum_i^N & \left( g_A^i - a_2 r_B^i(\lambda) + c_2 g_B^i + (1 - a_2 - c_2) \cdot b_B^i \right)^2 ; \\ \min_{a_3, c_3} \sum_i^N & \left( b_A^i(\lambda) - a_3 r_B^i(\lambda) + c_3 g_B^i(\lambda) + (1 - a_3 - c_3) \cdot b_B^i(\lambda) \right)^2 \end{aligned}$$

Решение задачи минимизации отдельно для каждого канала обеспечивает получение значений коэффициентов  $a_i, c_i, d_i$ , где  $i=1...3$  [6]. Это дает возможность рассчитать новые спектральные характеристики и определить на их основе матрицы для коррекции цвета. Перед расчётом новых спектральных характеристик необходимо выполнить процедуру баланса белого.

### 1.3.2. Матричная цветокоррекция

Для реализации цветовой калибровки может применяться другая методика, которая осуществляет пересчет цветового пространства камеры «А» в пространство камеры «В» с помощью матрицы линейного преобразования. Изображение, формируемое камерой «А» принимается как опорное, и реализуется пересчет координат цветов на изображении, формируемом камерой «В», так, чтобы их отличие от цветов опорного изображения было минимально.

Пусть  $O_i = (O\_R_i, O\_G_i, O\_B_i)$  и  $P_i = (P\_R_i, P\_G_i, P\_B_i)$  - векторы RGB-координат  $i$ -го цвета в изображениях О и Р, формируемых опорной и калибруемой камерой. Тогда преобразование одного цветового пространства

в другое может быть реализовано с помощью матрицы линейного преобразования, включающей 9 коэффициентов.

$$\mathbf{O}_i \approx \hat{\mathbf{O}}_i = \mathbf{P}_i \cdot \left[ A_{(3 \times 3)} \right],$$

$$A_{(3 \times 3)} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \\ \gamma_1 & \gamma_2 & \gamma_3 \end{bmatrix},$$

$$\alpha_1 + \beta_1 + \gamma_1 = 1; \alpha_2 + \beta_2 + \gamma_2 = 1; \alpha_3 + \beta_3 + \gamma_3 = 1; \quad (1)$$

Для определения параметров  $\alpha_i, \beta_i, \gamma_i, i=1..3$  необходимо минимизировать функцию ошибки, основанную на расстоянии между цветами опорного и калибруемого изображения:

$$\text{Func\_Err} = \sum_i^N \left[ \begin{aligned} & \left( VO\_R_i - (VP\_R_i + \alpha_2(VP\_G_i - VP\_R_i) + \alpha_3(VP\_B_i - VP\_R_i)) \right)^2 + \\ & \left( VO\_G_i - (VP\_G_i + \beta_1(VP\_R_i - VP\_G_i) + \beta_3(VP\_B_i - VP\_G_i)) \right)^2 + \\ & \left( VO\_B_i - (VP\_B_i + \gamma_1(VP\_R_i - VP\_B_i) + \gamma_2(VP\_G_i - VP\_B_i)) \right)^2 \end{aligned} \right],$$

где  $N$  - число цветов, по которым выполняется калибровка.

Минимизация сформированной функции ошибки

$$\min_{\alpha_2, \alpha_3, \beta_1, \beta_3, \gamma_1, \gamma_2} \text{Func\_Err}$$

позволяет определить коэффициенты  $\alpha_2, \alpha_3, \beta_1, \beta_3, \gamma_1, \gamma_2$ , коэффициенты  $\alpha_1, \beta_2, \gamma_3$  могут быть найдены из условия (1) необходимого для сохранения баланса белого.

Использование матрицы линейного преобразования с 12 коэффициентами позволяет учесть яркостные смещения в каналах цвета между опорным и калибруемым изображениями. Данная матрица образуется путем введения в вектор RGB-координат калибруемых цветов четвертой компоненты равной единице.

$$O = \begin{bmatrix} O\_R_1 & O\_G_1 & O\_B_1 \\ O\_R_2 & O\_G_2 & O\_B_2 \\ O\_R_N & O\_G_N & O\_B_N \end{bmatrix}; P = \begin{bmatrix} P\_R_1 & P\_G_1 & P\_B_1 \\ P\_R_2 & P\_G_2 & P\_B_2 \\ P\_R_N & P\_G_N & P\_B_N \end{bmatrix},$$

где  $O$  – матрица RGB-координат цветов опорного изображения;  $P$  - матрица RGB-координат цветов калибруемого изображения.

$$[I P] = \begin{bmatrix} 1 & P_{-R_1} & P_{-G_1} & P_{-B_1} \\ 1 & P_{-R_2} & P_{-G_2} & P_{-B_2} \\ 1 & P_{-R_N} & P_{-G_N} & P_{-B_N} \end{bmatrix}$$

Единичный вектор-столбец необходим для учета яркостных смещений между изображениями Р и О.

Поскольку, главной задачей является нахождение матрицы  $A_{(4 \times 3)}$ , которая наилучшим образом пересчитает цветовые координаты калибруемого изображения Р на основе координат опорного О, то преобразование цветовых пространств может быть представлено в матричной форме:

$$O \approx \hat{O} = [I P] A_{(4 \times 3)},$$

где  $\hat{O}$  - матрица RGB-координат цветов изображения Р после пересчета;  $A_{(4 \times 3)}$  – матрица линейного преобразования.

Таким образом, цвет  $i$ -го пикселя после преобразования будет представлять собой линейную комбинацию пересчитанных RGB-координат и постоянной смещения яркости. Например, координата красного цвета первого пикселя будет представлять собой:

$$\hat{O}_{-R_1} = A_{11} + A_{21} \cdot P_{-R_1} + A_{31} \cdot P_{-G_1} + A_{41} \cdot P_{-B_1}$$

Если количество независимых цветовых координат в опорном изображении превышает двенадцать, то для идентификации коэффициентов матрицы  $A_{(4 \times 3)}$ , можно использовать метод наименьших квадратов (МНК).

$$A_{(4 \times 3)} = \left( [I P]^T [I P] \right)^{-1} [I P]^T O;$$

$$A_{(4 \times 3)} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{bmatrix}.$$

Использование метода наименьших квадратов минимизирует сумму квадратов отклонений искомых цветовых координат от опорных и позволяет уменьшить сильные рассогласования в значениях между матрицами RGB-координат [7].



#### 1.4. Применение специальных мишеней для цветокоррекции

Проведение процедуры баланса белого является необходимостью при использовании метода сведения спектральных кривых или матрицы  $A_{(3 \times 3)}$ , состоящей из 9 коэффициентов. Поскольку при изменении условий освещения камера, в зависимости от цветового тона источника света, начинает по-другому воспроизводить белый цвет, что приводит к изменению координат цветов на формируемых изображениях. RGB-значения, полученные на выходе сенсора, зависят от его чувствительности и от спектра излучения источника освещения. Сенсоры разных камер имеют различные кривые чувствительности, зависящие как от состава полупроводника сенсора, так и от спектров пропускания примененных фильтров. Поэтому, на выходе сенсора белый цвет не всегда будет соответствовать значению 255 в каждом канале.

Для решения этой задачи реализуют процедуру баланса белого:

- с использованием специальной мишени;
- без использования мишени.

На данный момент существует множество алгоритмов автоматической настройки баланса белого. Чаще всего используют комбинацию двух методов:

1. Метод «серого мира» - Gray World Theory. Алгоритм основан на предположении, что средние значения красного, зеленого и синего каналов на участке наибольшей яркости должны быть равны  $R_{avg} = G_{avg} = B_{avg}$ . Этот метод берет свои корни из киносъемки, где среднее значение на негативах вместо того, чтобы быть нейтральным, было смещено в сторону темных областей [8].

Обозначим цветное изображение как  $L(x, y)$ , имеющее размер  $M \times N$ , где  $x$  и  $y$  соответствуют индексам положения пикселя, а  $L_r(x, y)$ ,  $L_g(x, y)$  и  $L_b(x, y)$ , соответственно, являются значениями RGB-каналов этого изображения. Вычисляются средние значения:

$$\begin{aligned} R_{avg} &= \frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N L_r(x, y); \\ G_{avg} &= \frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N L_g(x, y); \\ B_{avg} &= \frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N L_b(x, y). \end{aligned} \tag{2}$$

Если все три значения выражений (2) одинаковы, то изображение уже удовлетворяет условиям данного метода и не требует дальнейшей обработки.

В ином случае, зеленый канал берется в качестве опорного, и вычисляются коэффициенты усиления для красного и синего каналов:

$$\alpha = \frac{G_{avg}}{R_{avg}} \text{ и } \beta = \frac{G_{avg}}{B_{avg}} . \quad (3)$$

Новое изображение формируется через применение выражений (3) к красным и синим пикселям:

$$L_r(x, y) = \alpha L_r(x, y);$$

$$L_b(x, y) = \beta L_b(x, y).$$

2. Алгоритм «Ретинекс» - Retinex Theory. Данный метод основан на свойстве зрительной системы отождествлять белый цвет с максимальным откликом колбочек сетчатки глаза. Камера находит наиболее яркие пиксели (или вычисляет средние значения из суммы наиболее ярких пикселей)  $R_{max}$ ,  $G_{max}$ ,  $B_{max}$  и выравнивает значения во всех трёх каналах:

$$R_{max} = \max_{x,y} \{L_r(x, y)\};$$

$$G_{max} = \max_{x,y} \{L_g(x, y)\};$$

$$B_{max} = \max_{x,y} \{L_b(x, y)\}.$$

Оставляя канал В неизменным, коэффициенты усиления для красного и синего каналов будут равны:

$$\chi = \frac{G_{max}}{R_{max}} \text{ и } \delta = \frac{G_{max}}{B_{max}} . \quad (4)$$

Новое изображение формируется через применение выражений (4) к красным и синим пикселям:

$$I_r(x, y) = \chi I_r(x, y);$$

$$I_b(x, y) = \delta I_b(x, y).$$

Настройка баланса белого с использованием специальных мишеней представляет из себя настройку по нейтральному цвету в кадре. Нейтральный цвет — это цвет, не обладающий никакими оттенками и имеющий одинаковые значения красного, зеленого и синего в каждом канале.

На данный момент, применяется несколько видов специальных мишеней:

- серая карта;
- белая карта;
- шкала серого цвета;

- цветовая шкала.

Для установки эталонного цвета могут применяться серые или белые карты (рисунок 1). Если снимать карту при фиксированном освещении, то поканальный отклик будет произведением спектра осветителя и чувствительности сенсора камеры. Серая карта, включенная в снимок, учитывает цветовой тон и отраженный свет, так что при обработке изображения, если привести цветовой баланс к правильному отображению эталона, это позволит скорректировать остальные цвета на фотографии. Белая карта также позволяет настроить цветовую чувствительность камеры так, чтобы она точно соответствовала условиям освещения.



Рисунок 1 - Белая и 18% серая карты фирмы X-rite

На сегодняшний день, серая карта является наиболее используемой нежели белая. При производстве белых карт применяются разные отбеливатели и с течением времени карта может изменить свой оттенок, что в дальнейшем может привести к неверной цветокоррекции изображения.

#### 1.4.1. Виды серых шкал

Еще одним эффективным способом является использование специальных шкал серого цвета, содержащих некоторое количество градаций серого, либо цветовых шкал, содержащих определенный набор цветов. Фирма Kodak в своей шкале серого (рисунок 2) располагает 20 зон, которые соответствуют значениям оптической плотности от 0.05 до 1.95 с шагом 0.1. Шкала ColorChecker Gray Scale card фирмы X-rite (рисунок 3) состоит из трех полей - белого, 18% серого и черного. Она представляет собой однородную спектрально при любых условиях освещения поверхность, на которой расположены 3 удлиненных поля, в которых зафиксированы оптические стандарты нейтральных тонов: белое 90% (яркость цвета снижена на 10%),

серое 18% и черное 3,1%. Все три образца имеют абсолютно нейтральный тон. Каналы RGB имеют одинаковые значения, а параметры оттенка (H) и насыщенности (S) в цветовом пространстве HSL равны нулю [9].

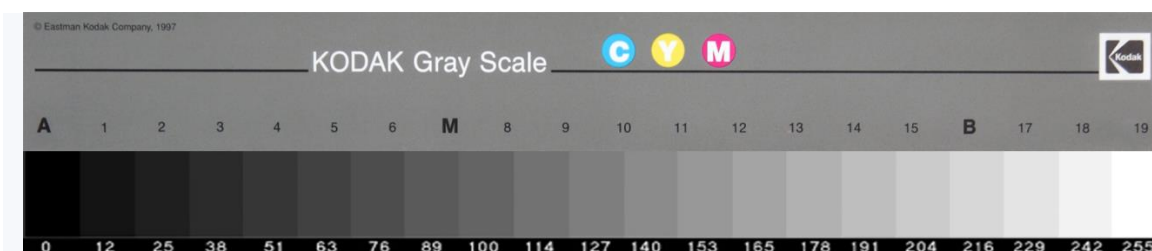


Рисунок 2 – Серая шкала фирмы Kodak



Рисунок 3 - Шкала ColorChecker Gray Scale card фирмы X-rite

#### 1.4.2. Виды цветовых шкал

Для реализации цветовой калибровки необходим набор тестовых цветов. Специальные цветовые шкалы позволяют получить на изображении все необходимые для калибровки цвета.

Наиболее распространенной и часто используемой цветовой шкалой является ColorChecker Classic, разработанная фирмой X-rite (рисунок 4). Стандартный набор цветов включает в себя 24 цветовых образца: линейку из шести ахроматических цветов (белый, черный, градации серого), первичные аддитивные цвета (Red, Green, Blue) и субтрактивные цвета (Cyan, Magenta, Yellow), а также дополнительные 12 цветов натуральных объектов, таких как

человеческая кожа (светлая, темная), голубое небо (синие оттенки), зелень ЛИСТВЫ И Т.Д.



| No. | Number              | sRGB |     |     | CIE L*a*b* |         |         | Munsell Notation   |            |
|-----|---------------------|------|-----|-----|------------|---------|---------|--------------------|------------|
|     |                     | R    | G   | B   | L*         | a*      | b*      | Hue Value / Chroma |            |
| 1.  | dark skin           | 115  | 82  | 68  | 37.986     | 13.555  | 14.059  | 3 YR               | 3.7 / 3.2  |
| 2.  | light skin          | 194  | 150 | 130 | 65.711     | 18.13   | 17.81   | 2.2 YR             | 6.47 / 4.1 |
| 3.  | blue sky            | 98   | 122 | 157 | 49.927     | -4.88   | -21.925 | 4.3 PB             | 4.95 / 5.5 |
| 4.  | foliage             | 87   | 108 | 67  | 43.139     | -13.095 | 21.905  | 6.7 GY             | 4.2 / 4.1  |
| 5.  | blue flower         | 133  | 128 | 177 | 55.112     | 8.844   | -25.399 | 9.7 PB             | 5.47 / 6.7 |
| 6.  | bluish green        | 103  | 189 | 170 | 70.719     | -33.397 | -0.199  | 2.5 BG             | 7 / 6      |
| 7.  | orange              | 214  | 126 | 44  | 62.661     | 36.067  | 57.096  | 5 YR               | 6 / 11     |
| 8.  | purplish blue       | 80   | 91  | 166 | 40.02      | 10.41   | -45.964 | 7.5 PB             | 4 / 10.7   |
| 9.  | moderate red        | 193  | 90  | 99  | 51.124     | 48.239  | 16.248  | 2.5 R              | 5 / 10     |
| 10. | purple              | 94   | 60  | 108 | 30.325     | 22.976  | -21.587 | 5 P                | 3 / 7      |
| 11. | yellow green        | 157  | 188 | 64  | 72.532     | -23.709 | 57.255  | 5 GY               | 7.1 / 9.1  |
| 12. | orange yellow       | 224  | 163 | 46  | 71.941     | 19.363  | 67.857  | 10 YR              | 7 / 10.5   |
| 13. | blue                | 56   | 61  | 150 | 28.778     | 14.179  | -50.297 | 7.5 PB             | 2.9 / 12.7 |
| 14. | green               | 70   | 148 | 73  | 55.261     | -38.342 | 31.37   | 0.25 G             | 5.4 / 8.65 |
| 15. | red                 | 175  | 54  | 60  | 42.101     | 53.378  | 28.19   | 5 R                | 4 / 12     |
| 16. | yellow              | 231  | 199 | 31  | 81.733     | 4.039   | 79.819  | 5 Y                | 8 / 11.1   |
| 17. | magenta             | 187  | 86  | 149 | 51.935     | 49.986  | -14.574 | 2.5 RP             | 5 / 12     |
| 18. | cyan                | 8    | 133 | 161 | 51.038     | -28.631 | -28.638 | 5 B                | 5 / 8      |
| 19. | white (.05*)        | 243  | 243 | 242 | 96.539     | -0.425  | 1.186   | N                  | 9.5 /      |
| 20. | neutral 8 (.23*)    | 200  | 200 | 200 | 81.257     | -0.638  | -0.335  | N                  | 8 /        |
| 21. | neutral 6.5 (.44*)  | 160  | 160 | 160 | 66.766     | -0.734  | -0.504  | N                  | 6.5 /      |
| 22. | neutral 5 (.70*)    | 122  | 122 | 121 | 50.867     | -0.153  | -0.27   | N                  | 5 /        |
| 23. | neutral 3.5 (1.05*) | 85   | 85  | 85  | 35.656     | -0.421  | -1.231  | N                  | 3.5 /      |
| 24. | black (1.50*)       | 52   | 52  | 52  | 20.461     | -0.079  | -0.973  | N                  | 2 /        |

Cie L\*a\*b\* values use Illuminant D50 2 degree observer sRGB values for Illuminate D65.  
© 2005, GretagMacbeth. All rights reserved.

Рисунок 4 - Цветовая шкала ColorChecker Classic фирмы X-rite

В связи с широким распространением цифровой фототехники фирма GretagMacbeth расширила стандартный ColorChecker добавлением дополнительных контрольных цветов для возможности использования в качестве цветовой мишени для профилирования цифровых фотокамер (эта версия мишени имеет обозначение ColorChecker DC), а позже еще раз усовершенствовала эту цветовую шкалу (версия ColorChecker Digital SG) (рисунок 5) [10].

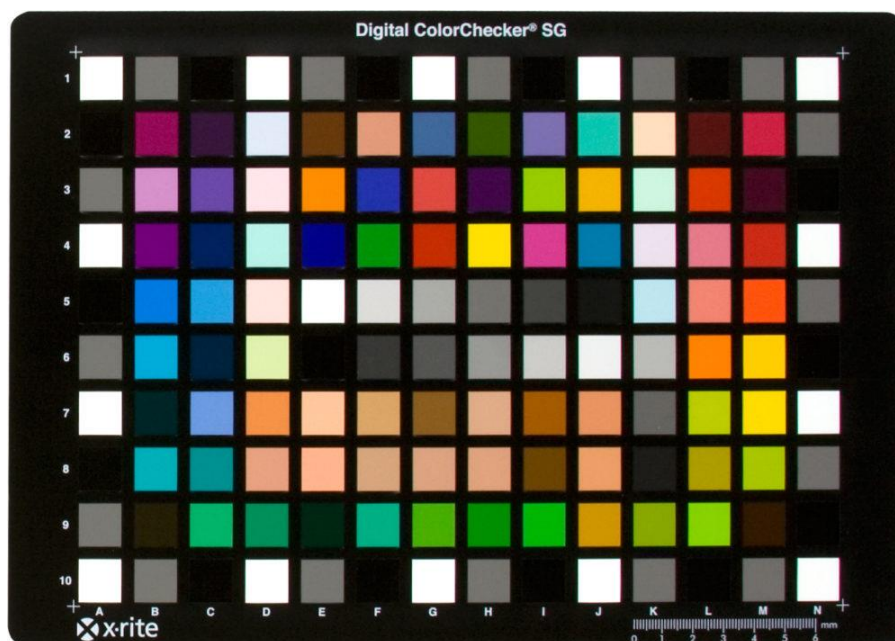


Рисунок 5 - Цветовая шкала ColorChecker Digital SG фирмы X-rite

### 1.5. Выводы и постановка задачи

Цветовая калибровка необходима для учета разных условий наблюдения, сенсоров с различными характеристиками и сопоставимости численных значений цветовых признаков в ПТС.

Особенность цветовой калибровки для медицинских систем — максимально уменьшить ошибку у определенных цветов, используемых далее в анализе. Недостатком метода синтеза «новых» спектральных характеристик является то, что ошибка минимизируется во всей спектральной области, то есть происходит усреднение ошибки. Человеческий глаз может и не различить ошибки цветопередачи. Поэтому, для наблюдателя ошибка в цветах может показаться не критичной. Как показали опыты Мак-Адама, человек не замечает изменения цвета вблизи некоторых порогов цветоразличения. Множества едва отличимых цветов аппроксимированы эллипсами на цветовой диаграмме, которые были названы эллипсами Мак-Адама или пороговыми. В автоматических системах присутствует информация о конкретных цветностях любой точки внутри данных областей. Для автоматических систем, которые выполняют классификацию и сегментацию по цветовому признаку, данная ошибка может привести к неверным результатам принадлежности признака к определенному объекту или классу.

При цветовой калибровке в ПТС медицинского назначения нет необходимости обеспечивать правильную передачу цветов во всех

спектральных диапазонах. Наиболее часто основой автоматического анализа являются конкретные цвета, и именно для них должны быть обеспечены сопоставимые количественные оценки. Это не может быть достигнуто методом, основанном на пересчёте спектральных кривых. Таким образом, для разработки процедуры калибровки следует взять матричный пересчёт. Чтобы не выделять, как отдельный шаг, баланс белого и, тем самым, упростить процедуру калибровки, целесообразно использовать матрицу преобразования цветовых пространств, состоящую из 12 коэффициентов. В этом случае, главной задачей является адаптация метода матричного пересчёта так, чтобы он обеспечивал минимум ошибки на строго определенных цветах. Это можно осуществить синтезированием нового тестового набора цветов или через присвоение цветам весовых коэффициентов.

## 2. РАЗРАБОТКА ПРОЦЕДУРЫ ЦВЕТОВОЙ КАЛИБРОВКИ ДЛЯ МУЛЬТИСПЕКТРАЛЬНОЙ СИСТЕМЫ МЕДИЦИНСКОЙ ДИАГНОСТИКИ

### 2.1. Цветовая калибровка на основе матрицы преобразования цветовых пространств

Под задачей калибровки понимают процедуру пересчета цветового пространства опорной камеры А под цветное пространство калибруемой камеры В, так чтобы ошибки в передаче цветов последней камерой были минимальны. Проведение процедуры цветокоррекции минимизирует ошибки

воспроизведения цветов каждой камерой, однако, не гарантирует, что калибруемая камера воспроизведёт цвета идентично опорной. Калибровка камеры В по камере А позволяет минимизировать ошибки цветопередачи именно между этими двумя камерами. В первой главе были рассмотрены несколько методов, на основе которых может выполняться калибровка. За основу процедуры калибровки, предложенной в данной работе, взят метод матричного пересчёта цветовых пространств. При реализации матричного пересчёта был использован метод наименьших квадратов, широко применяемый во многих областях обработки и анализа информации. Метод наименьших квадратов позволяет уменьшить рассогласования величин, за счет нахождения вектора параметров, при котором достигается минимум среднего квадрата ошибки. В данном случае, задача заключается в минимизации евклидова расстояния  $\left| \mathbf{O}_i - \hat{\mathbf{O}}_i \right|$  между двумя векторами — вектором восстановленных значений переменной и вектором фактических значений переменной. Необходимо найти некий вектор  $\mathbf{A}$ , минимизирующий ошибку

$$E_i = \left| \mathbf{O}_i - \hat{\mathbf{O}}_i \right|^2,$$

которая есть расстояние между координатами опорного цвета и соответствующего ему цвета в изображении Р после пересчета. Вектор  $\hat{\mathbf{O}}_i$  лежит в пространстве столбцов матрицы  $[I P]$ , поскольку  $\hat{\mathbf{O}}_i$  является линейной комбинацией матрицы  $[I P]$  с коэффициентами вектора  $\mathbf{A}_1, \dots, \mathbf{A}_i$ . Нахождение решения  $\mathbf{A}$  по методу наименьших квадратов эквивалентно задаче отыскания некой точки  $\hat{\mathbf{O}}_{R_i}$ , которая лежит ближе всего к опорному значению  $\mathbf{O}_{R_i}$  и при этом находится в пространстве столбцов матрицы  $[I P]$ .

Решением по методу наименьших квадратов несовместной системы  $\hat{\mathbf{O}} = \mathbf{O}$  будет уравнение:

$$[I P]^T \hat{\mathbf{O}} = [I P]^T \mathbf{O}.$$

Если столбцы матрицы  $[I P]$  линейно независимы, то матрица  $[I P]^T [I P]$  обратима и имеет единственное решение:

$$\mathbf{A}_{(4 \times 3)} = \left( [I P]^T [I P] \right)^{-1} [I P]^T \mathbf{O}.$$



## 2.2. Идентификация коэффициентов матрицы цветовой калибровки

Существенную ошибку при определении коэффициентов калибровки вносят цвета, имеющие значительные рассогласования RGB-координат в опорном и пересчитываемом изображениях. В этом случае целесообразно использовать итерационный метод наименьших квадратов с весовыми коэффициентами, который позволяет минимизировать влияние «выбросов» [6]. Особенностью данного подхода является использование весовой матрицы. Основная задача заключается в уменьшении весов выбросов в опорном  $O$  и пересчитанном  $\hat{O}$  изображениях, используя метрику Евклида или обратно ей пропорциональную весовую функцию.

Итерационный МНК позволяет на каждой следующей итерации получать приближенные значения искомых оценок параметров, лежащие ближе к истинному значению, чем значения предыдущей итерации. Используя обычный метод наименьших квадратов, определяются оценки коэффициентов матрицы  $A_{(4 \times 3)}$ , которые используются как начальное приближение. Чем ближе будет начальное приближение к окончательному решению, тем меньше потребуется итераций. Остановка процесса производится при достижении определенной точности оценок приближенных решений. В качестве критерия прерывания итераций используется стандартное условие  $\rho(x_{N+1}, x_N) < \varepsilon$ , где  $\rho(\dots)$  – некая метрика, а  $\varepsilon$  – параметр, определяющий точность решения. Как было сказано выше, в данной работе используется метрика Евклида:

$$dist(P, P') = \sqrt{\sum_{i=0}^N (A_i - B_i)^2},$$

где  $dist(P, P')$  – расстояние между объектами  $P$  и  $P'$ ;  $A_i$  – числовое значение  $i$ -й переменной для объекта  $P$ ;  $B_i$  – числовое значение  $i$ -й переменной для объекта  $P'$ ;  $n$  – число переменных, которыми описываются объекты или количество измерений.

Данная метрика позволяет определить расстояние между двумя точками  $\hat{O} - R_i$  и  $O - R_i$ . Чем больше значение  $dist(O - R_i, \hat{O} - R_i)$ , тем больше рассогласование между координатами красного цвета. Таким образом, зная значения RGB-координат цветового образца, мы можем сформировать вектор ошибки, который является мерой рассогласования между опорным и калибруемым цветом:

$$E_i = \sqrt{(O_{-R_i} - \hat{O}_{-R_i})^2 + (O_{-G_i} - \hat{O}_{-G_i})^2 + (O_{-B_i} - \hat{O}_{-B_i})^2},$$

где  $O_{-R_i}, O_{-G_i}, O_{-B_i}$  - координаты красного, зеленого и синего цветов  $i$ -го пикселя опорного изображения;  $\hat{O}_{-R_i}, \hat{O}_{-G_i}, \hat{O}_{-B_i}$  - координаты красного, зеленого и синего цветов  $i$ -го пикселя изображения Р после пересчета.

На основе вектора ошибок  $E_i$  формируется вектор весов:

$$C_i = \frac{1}{E_i + \varepsilon},$$

где  $\varepsilon$  - константа необходимая для исключения ситуации деления на ноль.

При малой размерности пространства возникает необходимость в нормализации вектора весов. Процедура нормировки вектора весов подразумевает под собой нормализацию значения каждого веса относительно всех элементов вектора. Чтобы получить нормированный вектор необходимо каждый элемент вектора разделить на евклидову норму

$$|C| = \sqrt{\sum_{i=1}^N C_i^2};$$

$$C_{i\text{норм}} = \frac{C_i}{|C|}.$$

Если ошибки измерений взаимно независимы, то формируется весовая матрица  $[C]$ , которая имеет диагональный вид. Элементами главной диагонали являются значения элементов нормированного вектора весов, возведенные в квадрат  $C_i^2$ .

$$C_{(N \times N)} = \begin{bmatrix} C_1^2 & 0 & 0 \\ 0 & C_2^2 & 0 \\ 0 & 0 & C_N^2 \end{bmatrix}.$$

Задача итерационной подстройки матрицы весов заключается в последовательной минимизации ошибки за счет корректировки весовых коэффициентов. На каждой итерации находятся оценки коэффициентов матрицы  $A_{(4 \times 3)}$  методом наименьших квадратов с учетом весовой матрицы:

$$A_{(4 \times 3)} = \left( [I \ P]^T C^2 [I \ P] \right)^{-1} [I \ P]^T C^2 O.$$

Эти оценки сравниваются с оценками коэффициентов матрицы  $A_{(4 \times 3)}$ , полученными на предыдущей итерации. Для нахождения оптимальных весов

необходимо условие, определяющее точность решения. Поэтому введем параметр  $\text{Err}$ , который будет необходимым условием точности оценки элементов матрицы  $A_{(4 \times 3)}$ . Если задать параметр  $\text{Err}$  равным 0.001, то прерывание итераций произойдет, когда элементы матриц сойдутся до 4 знака после запятой, то есть будет найдена оптимальная матрица весов.

Пошаговый алгоритм процедуры итерационного метода оценки коэффициентов матрицы  $A_{(4 \times 3)}$ :

Шаг 1: Определение оценки коэффициентов матрицы  $A_{(4 \times 3)}$ , используя обычный метод наименьших квадратов. Полученные оценки используются как начальное приближение.

Шаг 2: Формируют вектор ошибки сведения цветов  $E$ , где каждый элемент  $E_i$  равен расстоянию между координатами опорного цвета и соответствующего ему цвета в изображении  $P$  после пересчета, согласно метрике Евклида.

$$E_i = \sqrt{(O_{-R_i} - \hat{O}_{-R_i})^2 + (O_{-G_i} - \hat{O}_{-G_i})^2 + (O_{-B_i} - \hat{O}_{-B_i})^2}.$$

Шаг 3: Расчет вектора весов  $C$  с элементами:

$$C_i = \frac{1}{E_i + \varepsilon}.$$

Рекомендуемое значение  $\varepsilon = 0,1$

Шаг 4: Выполняют процедуру нормализации вектора весов  $C$  - каждый элемент вектора  $C$  делится на квадратный корень из суммы квадратов всех элементов вектора  $C$ .

Шаг 5: Формируют весовую матрицу, которая представляет собой диагональную матрицу, где элементами главной диагонали являются значения элементов нормированного вектора весов, возведенные в квадрат.

$$C_{(N \times N)} = \begin{bmatrix} C_1^2 & 0 & 0 \\ 0 & C_2^2 & 0 \\ 0 & 0 & C_N^2 \end{bmatrix}.$$

Шаг 6: Находят оценки коэффициентов матрицы  $A_{(4 \times 3)}$  методом наименьших квадратов с учетом весовой матрицы:

$$A_{(4 \times 3)} = ([I \ P]^T C^2 [I \ P])^{-1} [I \ P]^T C^2 O.$$

Шаги с 2 по 6 повторяют до необходимой точности оценки элементов матрицы  $A_{(4 \times 3)}$ .

## 2.3. Процедура цветовой калибровки с учётом степени значимости цвета

### 2.3.1. Процедура формирования весовых коэффициентов

Поскольку, цветовым выбросам присваиваются наименьшие веса, то задача итерационного алгоритма - увеличить вес выбросов, чтобы они оказывали меньшее влияние. Однако, цвета с наименьшими рассогласованиями, и, следовательно, имеющие наибольшие веса, не всегда являются приоритетными. Поэтому возникает необходимость в указании программе наиболее важных цветов, в которых в первую очередь должны быть минимизированы рассогласования координат.

Присвоение отдельных значений весов каждому цветовому образцу позволяет решить данную проблему, поскольку, каждое значение вектора  $C$  отвечает за величину рассогласования цветовых координат опорного и калибруемого изображений. Если калибровка выполняется по всем образцам стандартной мишени ColorChecker, то цветовым признакам, на основе которых осуществляется классификация, должны быть присвоены максимальные весовые коэффициенты. При нахождении матрицы линейного преобразования данный метод позволит уменьшить рассогласования координат в цветах наиболее критичных для дальнейшей обработки изображения.

Поскольку новый вектор весов должен быть сформирован с учетом присвоения наиболее значимым цветам наибольших весовых коэффициентов, следовательно, веса должны быть распределены внутри вектора неравномерно. Если поставить условие, что наиболее важные цветовые образцы должны быть выбраны в первую очередь, то максимальные веса должны находиться вначале вектора. Для выполнения данной задачи можно воспользоваться законом нормального распределения.

Нормальное распределение вероятностей играет важнейшую роль во многих областях знаний. Случайная величина подчиняется нормальному закону распределения, когда она подвержена влиянию большого числа случайных факторов. В общем случае, нормальное распределение зависит от среднего значения (математического ожидания) величины и дисперсии,

которая характеризует степень разброса значений величины относительно среднего. Нормальный закон распределения характеризуется плотностью вероятности вида:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-m)^2}{2\sigma^2}},$$

где  $m$ ,  $\sigma$  – фиксированные параметры,  $m$  – среднее,  $\sigma$  – стандартное отклонение.

Кривая распределения по нормальному закону имеет симметричный холмообразный вид (рисунок 6).

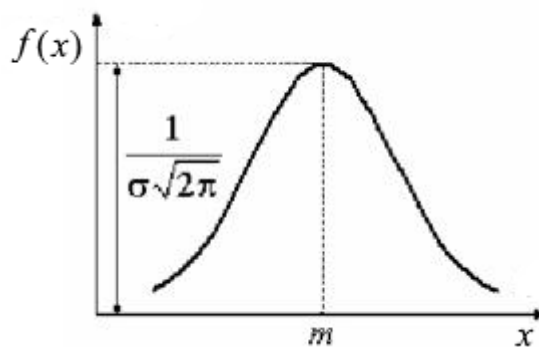


Рисунок 6 – Кривая распределения нормального закона [11]

Наиболее вероятные значения случайной величины расположены вблизи среднего значения  $m$ , где функция  $f(x)$  достигает максимума. Если изменять значение  $m$ , то кривая распределения будет смещаться вдоль оси абсцисс, без изменения своей формы. По мере удаления от среднего, вероятность значений уменьшается. Если значение попало за область  $m+3\sigma$ , то оно очень маловероятно (рисунок 7), потому что вероятность отклонения случайной величины от своего математического ожидания на величину, большую, чем  $3\sigma$ , практически равна нулю. На участке от  $m-\sigma$  до  $m+\sigma$  размещается в среднем 68% значений, в диапазоне от  $m-2\sigma$  до  $m+2\sigma$  размещается 95,45% всех значений, а на участке  $(m-3\sigma, m+3\sigma)$  – уже 99,73.

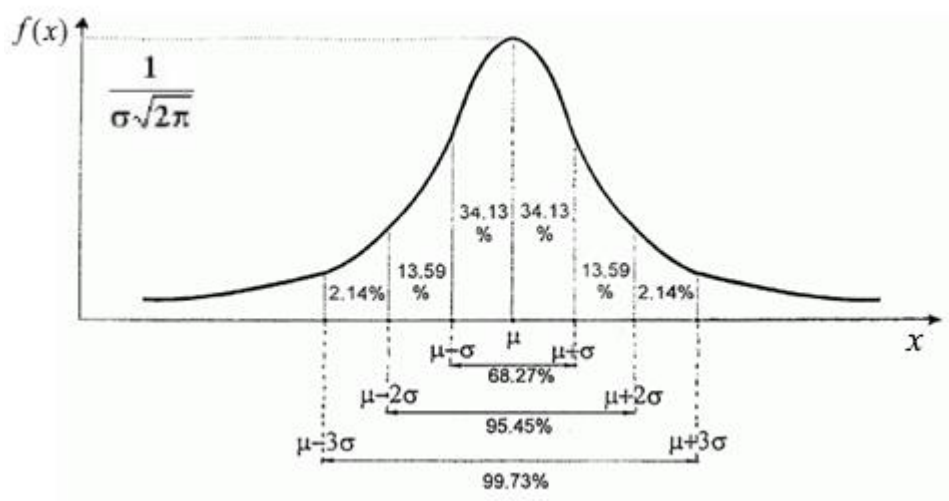


Рисунок 7 - График нормального закона распределения [12]

Поскольку площадь кривой распределения всегда должна оставаться равной единице, то при увеличении  $\sigma$  кривая плотности распределения становится более плоской, растягиваясь вдоль оси абсцисс. При уменьшении  $\sigma$  кривая распределения вытягивается вверх, сжимаясь к оси симметрии. На рисунке 8 приведены три нормальные кривые с математическим ожиданием  $m = 0$ . Кривая I соответствует самому большому значению  $\sigma$ , а кривая III – самому малому  $\sigma$ .

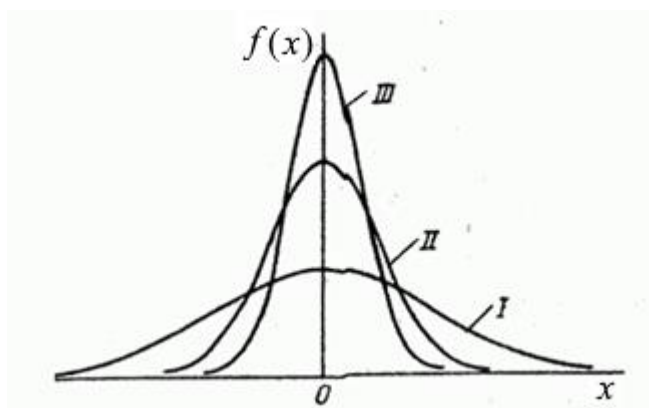


Рисунок 8 - Плотность вероятности нормального распределения [13]

### 2.3.2. Программная реализация процедуры формирования весовых коэффициентов

Для того, чтобы программно реализовать набор случайных величин необходимо воспользоваться генераторами псевдослучайных чисел (ГПСЧ), которые встроены в библиотеку `<random>` языка C++. Библиотека содержит в себе множество образов различных генераторов, поэтому выбор должен

осуществляться на основе поставленной задачи и доступных вычислительных ресурсов.

ГПСЧ генерируют псевдослучайные числа, используя параметр `seed` (ключ, зерно) – число, на основании которого происходит генерация. В случае повторного использования генератора с заданным параметром `seed` – на выходе получим идентичную численную последовательность, откуда и следует псевдослучайность генератора.

Для большинства задач оптимальным является генератор псевдослучайных чисел `std::mt19937`.

```
std::mt19937 gen ();
```

Он осуществляет генерацию чисел на основе алгоритма «вихрь Мерсенна» (mersenne twister). Данный алгоритм основывается на свойствах простых чисел Мерсенна. Числа Мерсенна - это числа вида

$$M_n = 2^n - 1,$$

где  $n$  - натуральное число.

Псевдослучайная последовательность чисел, порождаемая на выходе генератора, имеет большой период, равный числу Мерсенна, что обеспечивает быструю генерацию высококачественных по критерию случайности чисел. Вихрь Мерсенна обладает рядом преимуществ, в сравнении с другими генераторам псевдослучайных чисел, поскольку имеет большой период, меньшую предсказуемость и хорошие статистические свойства.

При формировании генератора случайных чисел (ГСЧ) параметр `seed` используется как источник энтропии. В общих словах, энтропия является мерой беспорядка, а информационная энтропия — мера неопределённости или непредсказуемости информации. Физическими источниками энтропии могут выступать тепловой шум компьютера или дробовой шум в резисторе. Некоторые UNIX-системы могут считывать шум с микрофона компьютера или с видеокамеры, используя разницу между двумя случайными кадрами. Таким образом, существует множество возможностей получить огромное количество случайных данных. В отличие от ГПСЧ, генераторы случайных величин формируют случайное число, предоставленное массивом данных различных источников энтропии.

В библиотеке `<random>` присутствует генератор `std::random_device`, работа которого основана на стохастическом процессе генерации недетерминированных случайных величин.

```
std::random_device rd;
```

Таким образом, в параметр `seed` генератора псевдослучайных чисел `std::mt19937` будет передаваться случайное число, созданное генератором случайных чисел `random_device`. На выходе, получим массив случайных величин с равномерным распределением. Библиотека `<random>` содержит набор классов распределений случайных величин, которые преобразовывают равномерное распределение генераторов в различные статистические распределения. Для моделирования нормального распределения используется класс `normal_distribution`.

```
std::normal_distribution<> distr();
```

`M`, `s` - параметры функции `distr`; `m` (mean) – математическое ожидание; параметр `s` (stddev) – отклонение  $\sigma$ . Эти параметры позволяют получить набор случайных величин, распределенных по нормальному закону, в заданном диапазоне.

Количество генерируемых величин определяется размерностью вектора весов, а размер вектора количеством цветовых образцов по которым производится калибровка. Сгенерированный набор чисел последовательно заносится в вектор.

Функция `sort` позволяет отсортировать значения коэффициентов вектора в порядке убывания.

```
sort(v.rbegin(), v.rend());
```

Чтобы привести вектор к единичному виду необходимо выполнить нормировку - каждый элемент вектора  $C$  делится на квадратный корень из суммы квадратов всех элементов вектора  $C$ . Формируется весовая матрица, где элементами главной диагонали являются значения нормированного вектора весов, возведенные в квадрат  $C_i^2$ .

$$C_{(N \times N)} = \begin{bmatrix} C_1^2 & 0 & 0 \\ 0 & C_2^2 & 0 \\ 0 & 0 & C_N^2 \end{bmatrix}.$$

Методом наименьших квадратов находятся коэффициенты матрицы линейного преобразования  $A_{(4 \times 3)}$ :

$$A_{(4 \times 3)} = \left( [I \ P]^T C^2 [I \ P] \right)^{-1} [I \ P]^T C^2 O.$$



Пошаговый алгоритм процедуры метода оценки коэффициентов матрицы  $A_{(4 \times 3)}$  с учетом весовых коэффициентов:

Шаг 1: Генерируется набор случайных величин, распределенных по нормальному закону.

Шаг 2: Формируется вектор весов  $C$ , куда заносится набор случайных величин.

Шаг 3: Выполняется сортировка коэффициентов вектора весов  $C$  по убыванию.

Шаг 4: Выполняется процедура нормализации вектора весов  $C$  - каждый элемент вектора  $C$  делится на квадратный корень из суммы квадратов всех элементов вектора  $C$ .

Шаг 5: Формируется весовая матрица  $[C]$ , которая представляет собой диагональную матрицу

$$C_{(N \times N)} = \begin{bmatrix} C_1^2 & 0 & 0 \\ 0 & C_2^2 & 0 \\ 0 & 0 & C_N^2 \end{bmatrix}.$$










Шаг 6: Находят коэффициенты матрицы  $A_{(4 \times 3)}$  методом наименьших квадратов с учетом весовой матрицы:

$$A_{(4 \times 3)} = \left( [I \ P]^T C^2 [I \ P] \right)^{-1} [I \ P]^T C^2 O.$$

Результат формирования весовых коэффициентов представлен в таблице 1. Первая строка содержит линейку тестового набора цветов в порядке их значимости. Первые два цветовых образца соответствуют цветам специальным флуоресцентных пластинок, используемых при тестировании мультиспектральных систем. Именно эти два цвета являются приоритетными, поэтому им присваиваются максимальные весовые коэффициенты, которые максимально минимизируют ошибки цветопередачи. Далее выбраны цвета синей и желтой гаммы, поскольку карта патологии строится на основе классификации флуоресцентных изображений по цветовым признакам канала  $b$  системы Lab. Во второй строке содержатся значения весовых коэффициентов, которые будут присвоены определенному цветовому образцу в результате работы предложенного алгоритма. При каждом запуске программы алгоритм рассчитывает новые значения весовых коэффициентов в зависимости от параметра  $seed$  и количества цветовых образцов.

Распределение весовых коэффициентов

Таблица 1

|                      |   |   |   |   |  |   |   |   |   |
|----------------------|---|---|---|---|--|---|---|---|---|
| Цветовой образец     |  |  |  |  |  |  |  |  |  |
| Весовые коэффициенты | 0,3286  | 0,3002  | 0,1756  | 0,0574  | 0,0508   | 0,0493  | 0,0324  | 0,0045  | 0,0012  |

### 3. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ ПРЕДЛОЖЕННОЙ ПРОЦЕДУРЫ ЦВЕТОВОЙ КАЛИБРОВКИ

#### 3.1. Мультиспектральные ПТС

В настоящее время существует значительный класс мультиспектральных информационно-измерительных систем, которые активно осваиваются в различных областях науки и техники. Существенное отличие данных систем в том, что они работают во всех диапазонах электромагнитного спектра. Это открывает уникальные возможности при использовании изображений, полученных мультиспектральными приборами.

Самым распространенным способом создания мультиспектральных систем до сих пор остается размещение в составе прибора нескольких рабочих каналов, каждый из которых работает в собственном диапазоне спектра. Сочетание нескольких спектральных каналов в оптико-электронных приборах при наличии вычислительных средств с большой скоростью обработки многомерной информации, получаемой от этих многоканальных систем, позволяет в реальном времени более надежно решать задачи обнаружения, распознавания, классификации и идентификации самых различных объектов [14].

На сегодняшний день в медицинской диагностике классификация и сегментация являются одними из наиболее важных и сложных задач. Мультиспектральные изображения предоставляют больше информации о свойствах объекта, чем спектры в отдельных точках или только монохромные

изображения, что позволяет достовернее выявлять патологию и следить за структурно-функциональными изменениями [15].

### 3.2. Мультиспектральная ТВ система диагностики рака шейки матки

#### 3.2.1. Общие сведения

Одним из наиболее распространенных видов злокачественных заболеваний у женщин является рак шейки матки. По данным Всемирной организации здравоохранения в мире ежегодно диагностируются свыше 500 000 новых случаев этого заболевания, половина случаев которого заканчивается летальным исходом [16]. Однако, при раннем обнаружении и соответствующем лечении, развитие рака может быть предотвращено. Но из-за бессимптомного течения ранних стадий заболевания женщины не обращаются вовремя за медицинской помощью. Поэтому с целью их выявления применяются массовые профилактические осмотры женщин – программы скрининга. Обычно при этом используется «пап-тест» (тест Папаниколау) – цитологическое исследование мазков, взятых с поверхности шейки матки и цервикального канала. Чувствительность пап-теста для случаев предрака и ранней стадии рака невелика – 75,6% [17], что приводит к пропуску около четверти пациентов, нуждающихся в лечении. Большую чувствительность обеспечивает кольпоскопическое обследование (70–97%). Кольпоскопическая диагностика базируется на визуальном распознавании и поэтому существенным образом зависит от квалификации врача. Инвазивный рак и поражения цервикального канала имеют собственные морфологические особенности, которые могут быть обнаружены в ходе клинического

визуального осмотра. Характеристики пораженных тканей, такие как цвет, либо помутнение, форма краев (границ) области эрозии, диаметр кровяных сосудов и расстояния между капиллярами считаются основными при вынесении диагноза.

В настоящее время в различных областях медицины и, в первую очередь, в онкологии, большое развитие получают методы диагностики, основанные на регистрации различий во флуоресцентном излучении патологических очагов и окружающих нормальных тканей, которые возникают при освещении поверхности ткани светом определенной длины волны ультрафиолетового излучения (УФ) или видимого диапазонов спектра [18], [19]. Для описания этих различий используется термин «флуоресцентный контраст». Если различия имеют эндогенное происхождение, то используют метод аутофлуоресцентной диагностики (АФД). В его основе — различия в интенсивности и спектральном составе эндогенного (аутофлуоресцентного) излучения нормальных и патологических тканей. Как показали клинические исследования, при возбуждении в УФ и синей области спектра интенсивность аутофлуоресцентного излучения в очагах рака намного меньше, чем в нормальных тканях (эффект «темного пятна»). Флуоресцентные методы диагностики ранних форм рака обладают крайне высокой чувствительностью, достигающей 90%, и существенно меньшей специфичностью [18], [19]. Глубина проникновения возбуждающего излучения крайне мала, поэтому позволяют выявлять патологические очаги, расположенные только в поверхностных слоях кожи или на слизистых оболочках органов. Длина волны возбуждающего излучения варьируется в зависимости от применяемого метода диагностики и используемого флюоресцентного маркера.

Основная задача мультиспектральной ТВ системы состоит в повышении эффективности кольпоскопического обследования. На основании кольпоскопических изображений, полученных в белом свете и свете флуоресценции система проводит количественную оценку степени патологии той или иной области, реализует дифференциальную диагностику. Применительно к задаче диагностики онкологических изменений на шейке матки — это означает возможность выявления различных состояний тканей.

Система медицинской диагностики рака шейки матки решает задачу классификации и анализа флуоресцентных изображений с последующим формированием дифференциальной карты патологии. Карта патологии

строится на основе классификации флуоресцентных изображений по цветовым признакам в (система Lab) и Cr,Cb (система YCrCb).



### 3.2.2. Аппаратная реализация мультиспектрального комплекса LuxCol

Опытный образец мультиспектрального комплекса для проведения клинических испытаний включает [16]: систему освещения, оптическую систему, детекторную систему и компьютер со специальным программным обеспечением для обработки, анализа и хранения данных (рисунки 9, 10).

Комплекс позволяет получить следующий набор кольпоскопических изображений: изображения в белом свете, в белом поляризованном свете, флуоресцентные изображения, полученные при возбуждающем излучении 360нм, 390нм, 430 нм и 390нм с лазерной подсветкой 650 нм. По результатам проведенных исследований, как наиболее эффективные, для диагностики использовались только флуоресцентные изображения, полученные при возбуждающем излучении 360 нм и 390 нм.

Система освещения состоит из лампового осветителя на ртутной лампе и лазера 635нм. Суммирование этих излучений, требуемое в режиме двухволнового освещения FL(390+635), производится непосредственно на объекте. Максимальная неравномерность освещения



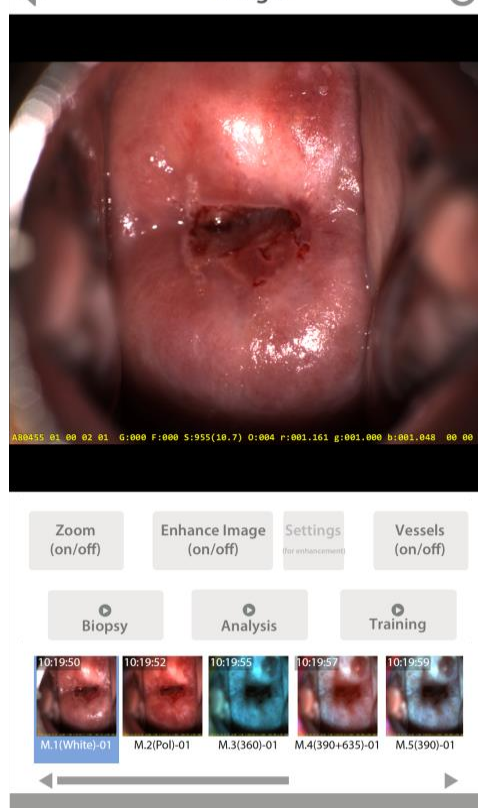
поля зрения не превышает  $\pm 5 \%$  в ламповом канале и  $\pm 15 \%$  - в лазерном канале.

Оптической системой является специальный проекционный объектив с фокусным расстоянием 100 мм и относительным отверстием F/2,8. Рабочее расстояние составляет в среднем 270 мм.

Опытный образец мультиспектрального комплекса для проведения клинических испытаний включает [16]: систему освещения, оптическую систему, детекторную систему и компьютер со специальным программным обеспечением для обработки, анализа и хранения данных (рисунки 9, 10).

Детекторная система состоит из цифровой видеокамеры и детекторных фильтров. Цифровая камера построена на базе ПЗС-приемника формата 2/3

Рисунок 10 - Вид  
детекторной системы



дюйма RGB типа с прогрессивным сканированием (ICX285AQ, SONY). Максимальная кадровая частота работы камеры составляет 15 Гц при формате кадра 1280x1024 элемента. Ее собственный шум составляет 10е, нелинейность характеристики свет-сигнал при перепадах освещенности в 100 раз не превышает 3,5%.

Сигналы из камеры поступают в компьютер по последовательной высокоскоростной шине, работающий по протоколу USB-2.0. Управление осветителем и лазерами осуществляется с компьютера через COM-порт. Вид рабочих окон специализированного программного обеспечения представлен на рисунке 11.

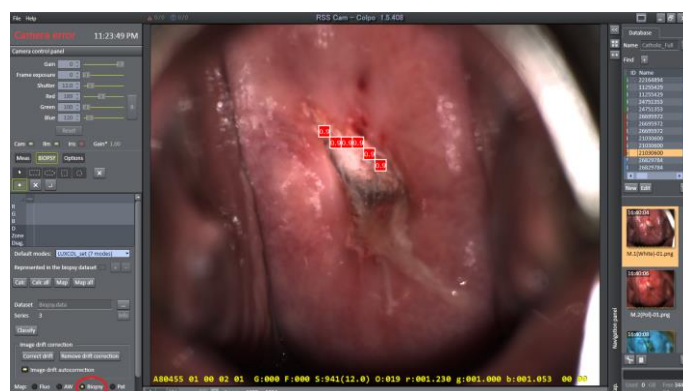


Рисунок 11 - Вид рабочих окон специализированного программного обеспечения для мобильной и стационарной версии комплекса

### 3.2.3. Обоснование необходимости калибровки

В коммерческих целях ТВ системы медицинской диагностики оснащаются камерами с КМОП-матрицами. В то время как спектральные характеристики ПЗС-матрицы несколько лучше, так как более приближены к кривым спектральной чувствительности глаза.

Характеристики спектральной чувствительности сенсора камеры с ПЗС-датчиком и камеры системы диагностики имеют существенные отличия в зеленом канале (рисунки 12, 13, 14). Это привело к тому, что в изображениях, полученных коммерческим экземпляром, слабо выражены цветовые признаки, являющиеся основой классификации: практически отсутствуют изменения цветового признака *b* (рисунки 12, б, 14).

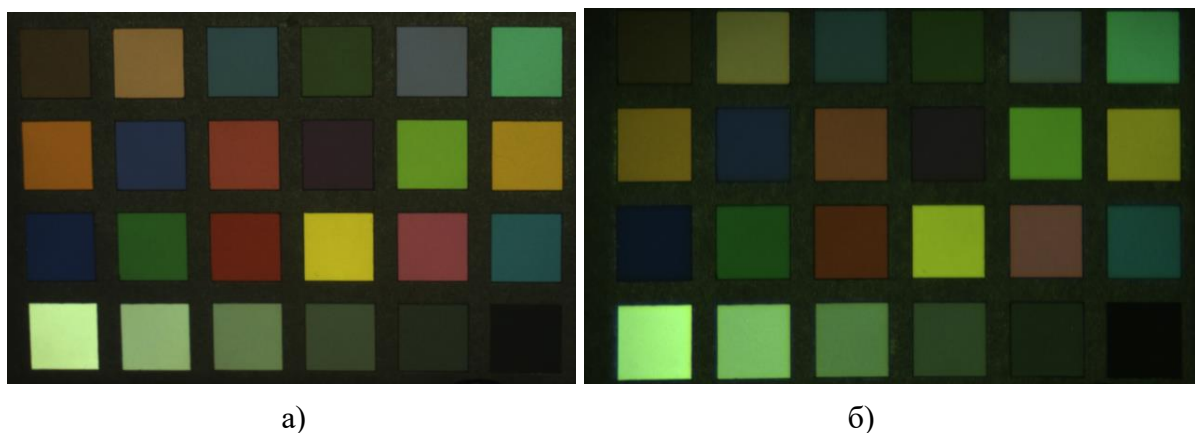


Рисунок 12 – Изображения тестовых таблиц

а) тестовая таблица, полученная с помощью ПЗС-датчика б) тестовая таблица, полученная с помощью КМОП-датчика

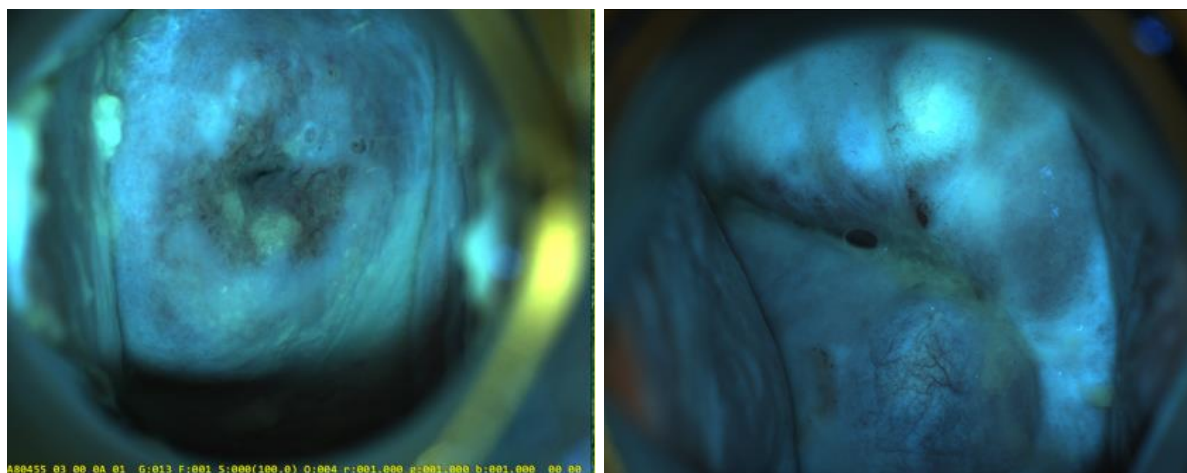


Рисунок 13 – Изображения шейки матки, полученные с помощью ПЗС-датчика. Длина волны возбуждающего излучения – 360 нм.



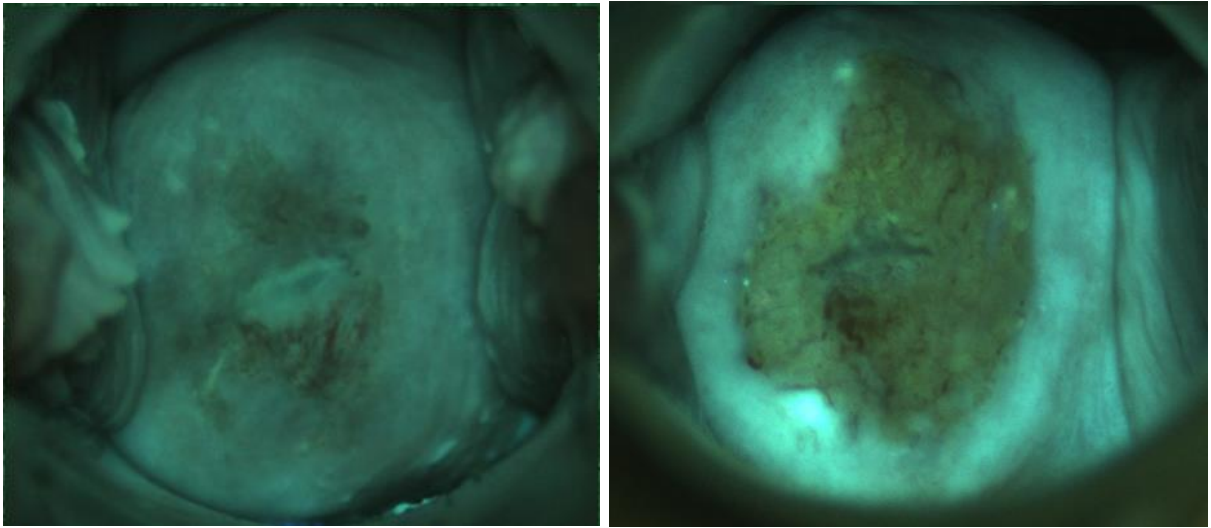


Рисунок 14 – Изображения шейки матки, полученные с помощью КМОП-датчика. Длина волны возбуждающего излучения – 365 нм.

Существенное отличие формируемых для анализа изображений не позволяет использовать классификатор, обученный на базе данных изображений, полученных ПЗС-сенсором, и значительно затрудняет классификацию по цветовому признаку в целом. Поэтому применение цветовой калибровки является необходимым условием при проведении анализа кольпоскопических изображений.

### 3.3. Методика эксперимента

#### 3.3.1. Описание моделирующего программного обеспечения

Программа реализована на языке объектно-ориентированного программирования C++ (Visual studio 2015) и осуществляет процедуру цветовой калибровки ТВ камер медицинского назначения за счет пересчета цветовых пространств камер с помощью матрицы линейного преобразования, состоящей из 12 коэффициентов. Программа имеет отдельный интерфейс (рисунок 15), с помощью которого осуществляется вывод тестовых изображений – тестовых таблиц ColorChecker, по которым осуществляется калибровка, а также вывод в отдельных окнах откалиброванных изображений.



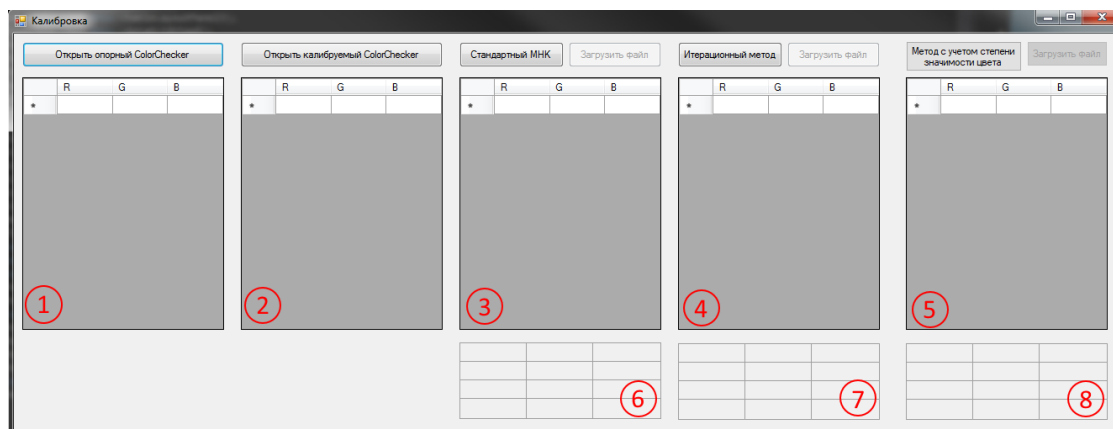


Рисунок 15 - Интерфейс программы

На форме программы расположены таблицы, содержащие:

- матрицы RGB-координат цветов опорного и калибруемого изображений (таблицы 1, 2);
- матрицы ошибок сведения цветов (таблицы 3, 4, 5);
- рассчитанные матрицы линейного преобразования (таблицы 6, 7, 8).

Как говорилось выше, особенностью данной процедуры является самостоятельный выбор набора цветов для калибровки системы. В настоящее время сертифицированные программные продукты, позволяющие проводить калибровку цветовых пространств изображений и создающий цветовые профили на основе этих данных, имеют функцию автоматического определения местоположения тестовых таблиц ColorChecker на фотографиях, а также считывания цветовых координат всех образцов шкалы. Однако, для изображений для которых калибровка цветовых признаков является основой для классификации и сегментации, должна быть возможность выбора именно тех цветов, для которых должна быть обеспечена максимально достоверная цветопередача. Поэтому, набор цветов для калибровки формируется пользователем самостоятельно, в зависимости от целей и задач дальнейшей работы с изображениями.

При нажатии на кнопку «Открыть опорный ColorChecker» на экран выводится опорное изображение, по которому выполняется калибровка (рисунок 16).

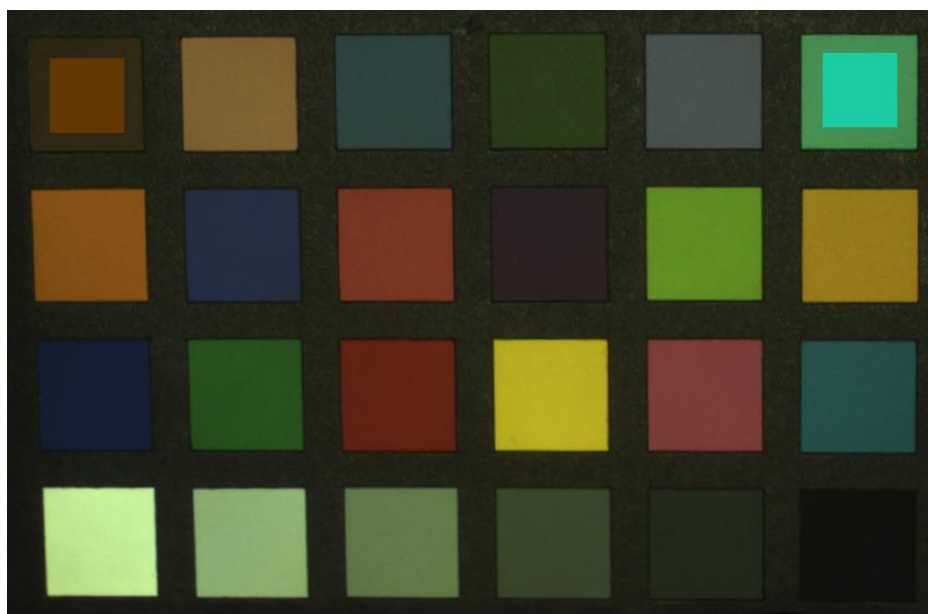


Рисунок 16 – Опорный ColorChecker

При нажатии на кнопку «Открыть калибруемый ColorChecker» на экран выводится калибруемое изображение (рисунок 17).

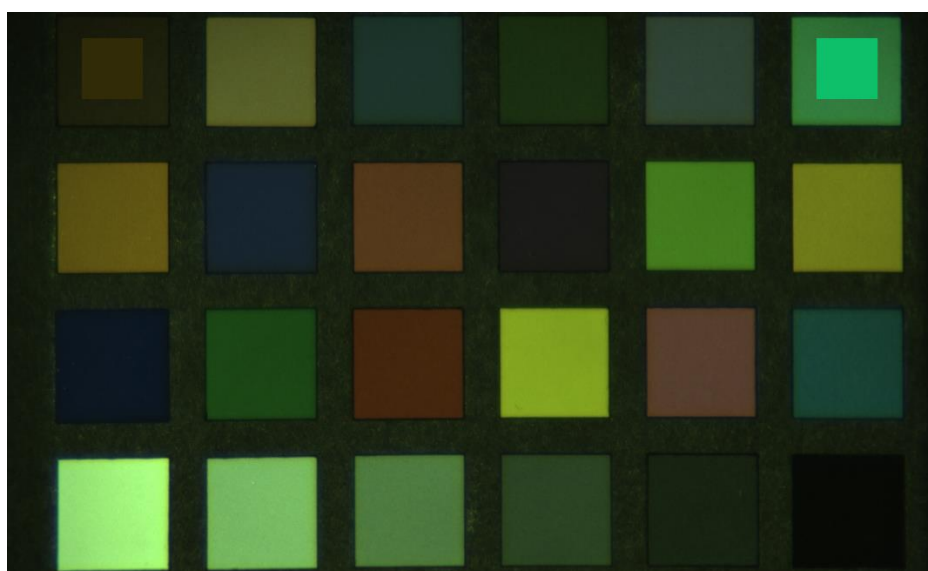


Рисунок 17 – Калибруемый ColorChecker

Пользователю предлагается сформировать набор тестовых образцов. Выбор цвета происходит нажатием кнопкой мыши на центр интересующего цветового образца. В левом верхнем углу экрана располагаются два небольших окна для быстрого просмотра выбранных на тестовой таблице образцов цвета (рисунок 18). Это дает возможность пользователю оценить визуальное рассогласование цветовых образцов.

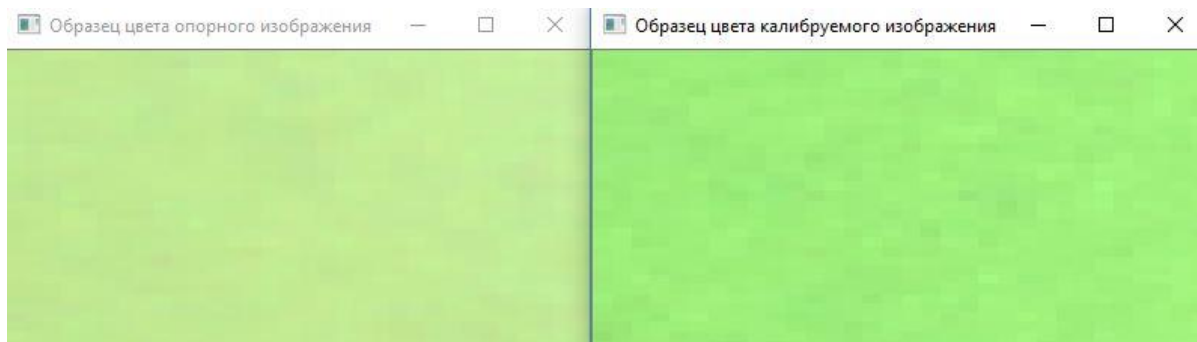


Рисунок 18 - Окна быстрого просмотра цветowych образцов

В процессе выбора пользователем цветowych образцов программа формирует два независимых массива цветowych координат опорного и калибруемого изображений. За счёт этого имеется возможность одновременно выбирать один и тот же образец на обоих изображениях.

В таблицу 1 на рисунке 19 (также см. рисунок 15) выводится массив RGB-координат сформированного на тестовом наборе опорного изображения. Каждая цветовая координата соответствует среднему значению по нескольким пикселям. Область усреднения имеет размер 25x25 пикселей с центром в позиции указателя мыши и необходима для исключения влияния пикселей с большими отклонениями в RGB-координатах.

В таблицу 2 выводится массив, сформированный по тем же правилам, только применительно к калибруемой таблице ColorChecker.

| Открыть опорный ColorChecker |            |            |            |
|------------------------------|------------|------------|------------|
|                              | R          | G          | B          |
| ▶                            | 27         | 205        | 162        |
|                              | 99         | 56         | 3          |
|                              | 20,2109... | 30,7011... | 51,5683... |
|                              | 151,856... | 122,775... | 25,7783... |
|                              | 36,1689... | 46,3144... | 65,5615... |
|                              | 132,614... | 80,1445... | 21,3916... |
|                              | 45,0126... | 67,9042... | 63,8837... |
|                              | 125,481... | 97,6386... | 49,8525... |
|                              | 36,5234... | 78,6748... | 73,9101... |
|                              | 190,775... | 230,487... | 143,902... |
|                              | 140,054... | 174,727... | 114,828... |
|                              | 97,5898    | 121,939    | 74,7607    |

| Открыть калибруемый ColorChecker |            |            |            |
|----------------------------------|------------|------------|------------|
|                                  | R          | G          | B          |
| ▶                                | 13         | 192        | 105        |
|                                  | 49         | 43         | 7          |
|                                  | 13,0214... | 23,7226... | 35,1337... |
|                                  | 110,139... | 122,147... | 25,0927... |
|                                  | 23,4267... | 36,9521... | 48,0078... |
|                                  | 93,6914... | 83,0507... | 18,9853... |
|                                  | 28,9238... | 53,2871... | 43,9042... |
|                                  | 85,7597... | 92,8623... | 40,2900... |
|                                  | 25,0224... | 66,7626... | 53,2392... |
|                                  | 154,983... | 235,021... | 120,667... |
|                                  | 121,138... | 187,663... | 99,9453... |
|                                  | 79,2832    | 123,301    | 64,0839    |

Рисунок 19 – Таблицы 1 и 2, заполненные значениями RGB-координат

После того как сформированы массивы RGB-координат производится вычисление матрицы линейного преобразования.

В программе реализованы три метода, с помощью которых можно осуществлять пересчет цветовых пространств:

- стандартный МНК;
- итерационный МНК с весовыми коэффициентами;
- МНК с весовой функцией.

На форму выведены три таблицы, в которых отображаются рассчитанные значения ошибок цветопередачи в каждой цветовой координате (рисунок в начале главы). Для определения величины этих ошибок производится вычисление разницы RGB-координат опорного и соответствующего ему образца в изображении Р после пересчета с использованием матрицы линейного преобразования.

Также на форму осуществляется вывод рассчитанных матриц линейного преобразования  $A_{(4 \times 3)}$  в таблицы 6, 7, 8 (рисунок 20). На рисунке показаны рассчитанные значения ошибок для каждого цветового канала и вычисленные матрицы  $A_{(4 \times 3)}$  с использованием трех различных методов. Набор тестовых образцов, на основе которых осуществлялось преобразование, соответствует линейке цветов, приведенной в таблице 2.

Стандартный МНК

Загрузить файл

|  | R           | G           | B           |
|--|-------------|-------------|-------------|
|  | 4.34594...  | 0.94813...  | 2.50398...  |
|  | 19.7199...  | 6.80882...  | -5.49040... |
|  | -1.53731... | -2.63074... | 2.05284...  |
|  | -4.65563... | 1.89855...  | 1.51640...  |
|  | 5.68854...  | 1.51984...  | 1.81500...  |
|  | -0.27963... | -3.07820... | 5.73110...  |
|  | 3.27704...  | 6.83538...  | 3.99999...  |
|  | 9.32255...  | 4.10901...  | 4.99986...  |
|  | 0.86066...  | 3.52512...  | -0.38388... |
|  | 3.85458...  | 4.38383...  | 1.81305...  |
|  | -8.21020... | -8.55724... | -5.28972... |
|  | -6.32706... | -2.90459... | -5.12904... |

Итерационный метод

Загрузить файл

|  | R           | G           | B           |
|--|-------------|-------------|-------------|
|  | 0.00057...  | -0.00014... | 0.00050...  |
|  | 24.8304...  | 7.87834...  | -7.21001... |
|  | -1.77339... | -0.00149... | 0.00210...  |
|  | 0.00045...  | 0.00112...  | 0.00043...  |
|  | 6.11637...  | 3.79136...  | -0.25014... |
|  | 4.47411...  | -3.71054... | 4.17733...  |
|  | 4.34685...  | 8.50687...  | 1.97637...  |
|  | 11.8227...  | 3.65460...  | 3.28874...  |
|  | 1.01765...  | 5.03221...  | -2.48437... |
|  | -0.00070... | -0.00023... | -8.95862... |
|  | -10.7188... | -11.3561... | -7.15968... |
|  | -6.14450... | -3.73242... | -7.02434... |

Метод с учетом степени значимости цвета

Загрузить файл

|  | R           | G           | B           |
|--|-------------|-------------|-------------|
|  | 0.01883...  | -0.01231... | 0.01168...  |
|  | 1.19339...  | 0.40783...  | -0.35903... |
|  | -6.88535... | -3.05583... | 1.54643...  |
|  | -10.3320... | -2.73462... | 1.99712...  |
|  | 9.94275...  | 3.95691...  | -2.38279... |
|  | -4.91065... | -6.53856... | 5.86864...  |
|  | 1.57878...  | 6.92884...  | 2.07899...  |
|  | 14.6271...  | 4.31893...  | 1.01052...  |
|  | 0.55284...  | 4.17105...  | -3.04781... |
|  | 52.5706...  | 16.2821...  | -19.2330... |
|  | 26.4483...  | -0.01620... | -21.0101... |
|  | 5.04327...  | -0.52288... | -11.9083... |

|          |          |          |
|----------|----------|----------|
| 17.93397 | 12.51659 | 8.04094  |
| 1.14838  | -0.15286 | -0.30234 |
| 0.1906   | 1.0352   | 0.16242  |
| -0.44575 | -0.04986 | 1.18286  |

|          |          |          |
|----------|----------|----------|
| 13.07016 | 9.51776  | 9.97378  |
| 1.13596  | -0.13532 | -0.30794 |
| 0.18189  | 1.06242  | 0.16303  |
| -0.34059 | -0.06423 | 1.18789  |

|          |          |          |
|----------|----------|----------|
| 47.37097 | 20.65345 | -0.86452 |
| 0.76506  | -0.24838 | -0.17071 |
| 0.49569  | 1.14581  | 0.05244  |
| -1.19532 | -0.30864 | 1.47622  |

Рисунок 20 - Таблицы ошибок цветопередачи и рассчитанные матрицы линейного преобразования

Теперь есть возможность применить полученную матрицу цветового преобразования к изображению и, тем самым, осуществить пересчет его

цветовых координат. Для выбора изображения на форме присутствует кнопка «Загрузить файл». На форме присутствуют три кнопки загрузки файла и каждая из них относится к определенному методу. Каждая из них будет доступна только после вычисления матрицы преобразования с использованием, привязанного к ней, метода.

При нажатии на кнопку открывается окно обозревателя файлов и папок компьютера, в котором пользователь может выбрать интересующее его изображение.

После выбора необходимого файла, программа автоматически осуществит пересчёт цветового пространства выбранного изображения с учетом матрицы линейного преобразования. На форму будут выведены два окна, содержащие в себе оригинальное изображение и пересчитанное (рисунок 21).

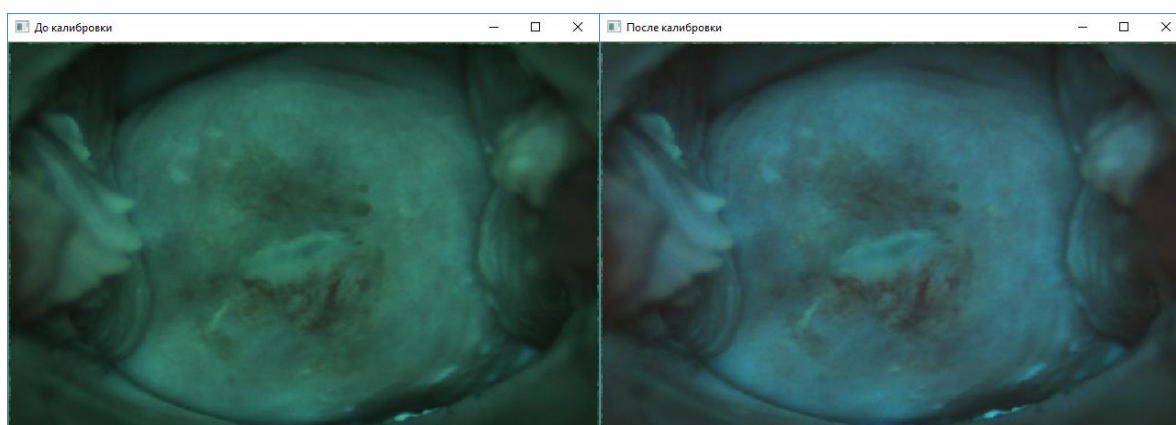


Рисунок 21 – Вид окон «До калибровки» и «После калибровки»

Вывод изображений в отдельные окна позволяет пользователю манипулировать высотой и шириной окна, а также переносить его на дополнительные мониторы для удобства работы.

### 3.3.2. Критерии оценки качества цветовой калибровки

Главной задачей при анализе результатов является оценка эффективности восстановления информации о цветовой координате  $b$  в формируемых изображениях. Это необходимо для обеспечения корректного решения задачи классификации для пересчитанных флуоресцентных кольпоскопических изображений. Как упоминалось выше, карта патологии строится на основе классификации флуоресцентных изображений по цветовым признакам  $b$  системы Lab и Cr, Cb системы YCrCb.



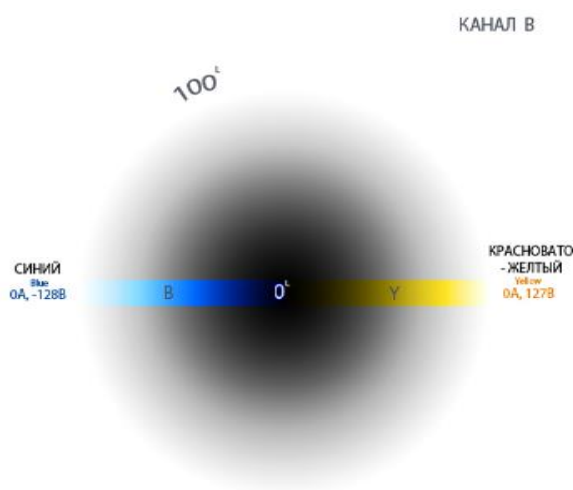


Рисунок 22 – Цветовое пространство Lab

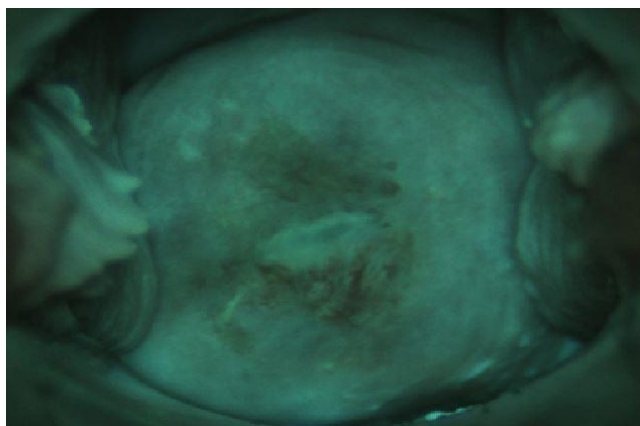
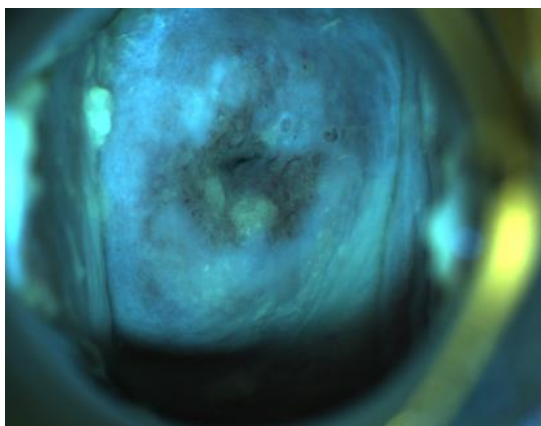
В модели Lab (рисунок 22) значения хроматической составляющей цвета (тон, насыщенность) отделены от светлоты. Светлота задана координатой L (изменяется от 0 до 100), хроматическая составляющая — двумя декартовыми координатами а и b. Координата а охватывает цвета от пурпурного цвета до зеленого. Координата b отвечает за палитру от синего до желтого цвета. В зоне нулевых значений каналов а и b расположены ахроматические цвета

(близкие к серому) (рисунок 23).

Система Lab нашла широкое применение в программном обеспечении для обработки изображений в качестве промежуточного цветового пространства, через которое происходит конвертирование данных между другими цветовыми пространствами. При этом особые свойства Lab сделали редактирование в этом пространстве мощным инструментом цветокоррекции [20].

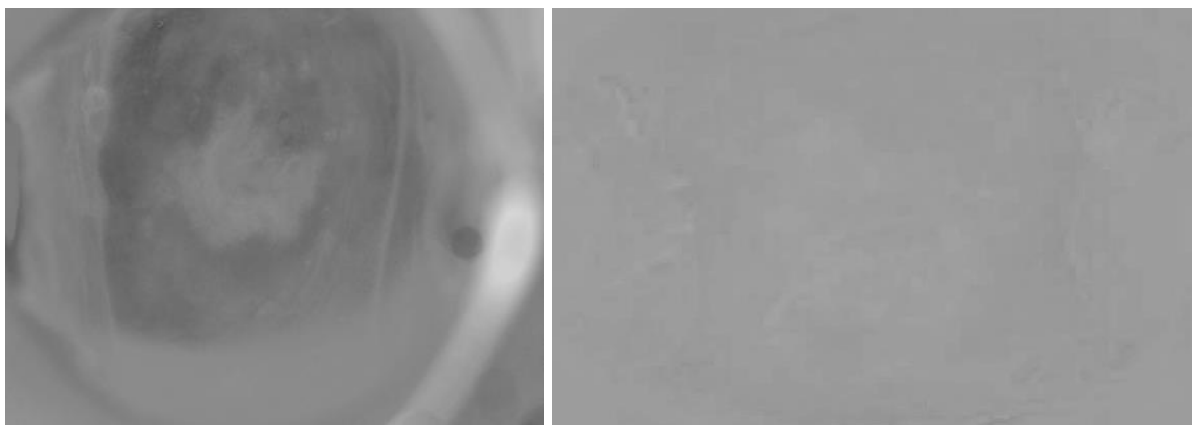
Как можно видеть на рисунке 24, на изображении полученным коммерческим экземпляром, слабо выражены цветовые признаки, являющиеся основой классификации - практически отсутствуют изменения цветового признака b.

Рисунок 23 – Графическое отображение канала b системы Lab



а)

б)



в)

г)

Рисунок 24 – Изображения, полученные разными сенсорами и их координаты  $b$   
а, б) - изображения, полученные сенсорами ПЗС 285 и КМОП 236; в, г) - координата  $b$  в  
изображениях

Поскольку, медицинские изображения содержат ограниченное количество цветов, необходимо синтезировать новый тестовый набор цветовых образцов, по которым будет осуществлена цветокоррекция изображений. Набор цветов для калибровки был сформирован на основе стандартного ColorChecker, из которого были выбраны образцы, позволяющие минимизировать ошибку при передаче цветов синей и желтой гаммы. Именно эти цвета присутствуют на флуоресцентных кольпоскопических изображениях, полученных при возбуждающем излучении с длиной волны 360 нм, и являются основой классификации.

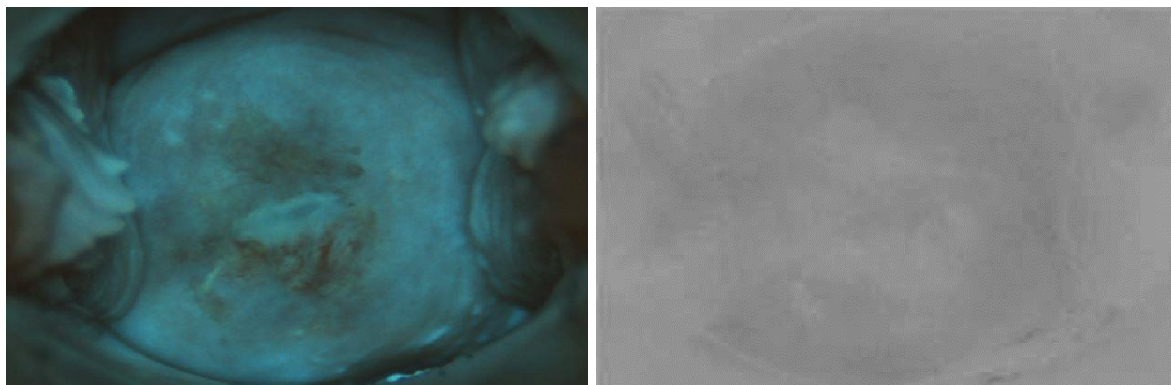
Сформированный набор включает 9 цветовых образцов и линейку градаций серого (рисунок 25). К изображениям не была применена процедура баланса белого, поэтому использование серой шкалы в тестовой таблице является обязательным.



Рисунок 25 – Сформированный набор цветовых образцов

Так как при использовании метода с использованием весовой функции необходимо выбирать цвета в определенном порядке, цвета синей и желтой гаммы были выбраны в первую очередь как наиболее важные для калибровки.

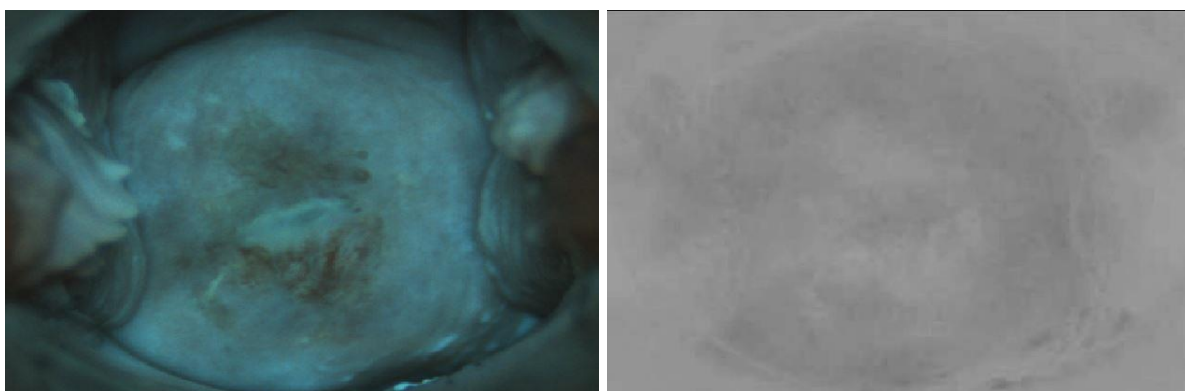
На нижеприведенных рисунках 26, 27, 28 представлены результаты работы программы с использованием трех методов.



а)

б)

Рисунок 26 – Результат обработки изображения с использованием МНК  
а) изображение после использования процедуры калибровки; б) - координата  $b$  этого изображения



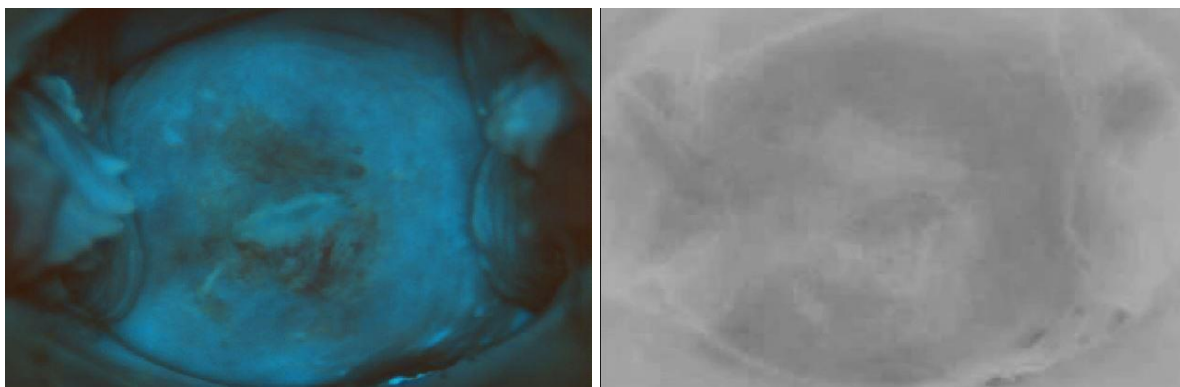
а)

б)

Рисунок 27 – Результат обработки изображения с использованием итерационного МНК

а) изображение после использования процедуры калибровки на основе МНК;  
б) - координата  $b$  этого изображения





а)

б)

Рисунок 28 – Результат обработки изображения с использованием предложенного метода

а) изображение после использования процедуры калибровки на основе; б) - координата  $b$  этого изображения

После применения процедур калибровки к изображению, полученному коммерческим экземпляром, происходит заметное восстановление информации о цветовой координате  $b$ . Как можно видеть на рисунке, метод с использованием весовой функции обеспечивает более эффективное восстановление необходимых цветов, по сравнению с другими методами.

### 3.4. Анализ результатов экспериментального исследования

Основными задачами предложенного метода калибровки являлось уменьшение рассогласования цветовых координат опорного и калибруемого изображений и обеспечение минимума ошибки цветопередачи на строго определенных цветах. Величина вектора ошибки должна быть минимальна для данных цветов, в то время как, цвета не играющие важную роль в дальнейшей классификации могут иметь большие рассогласования, поскольку, в рамках поставленной задачи, необходимо обеспечить максимальное восстановление информации о координате  $b$  изображения.

Результаты работы реализованных методов калибровки представлены в таблице 2. Первая строка содержит линейку тестового набора цветов в порядке их значимости. Именно первые два цветовых образца являются приоритетными, так как они соответствуют цветам флуоресцентных пластинок, входящих в тесты для мультиспектральных систем. Поэтому, данным цветам присваиваются максимальные весовые коэффициенты, которые максимально минимизируют ошибки цветопередачи.

Далее выбраны цвета синей и желтой гаммы, поскольку карта патологии строится на основе классификации флуоресцентных изображений по цветовым признакам канала *b* системы Lab. Линейка градаций серого необходима для процедуры баланса белого.

Столбцы  $\Delta E$  содержат значения ошибок сведения цветов. Значения  $\Delta E_{\text{МНК}}$  соответствуют ошибкам при использовании метода наименьших квадратов,  $\Delta E_{\text{итер}}$  – итерационного МНК с весовыми коэффициентами и  $\Delta E_{\text{степень знач.}}$  – метода с учетом степени значимости цвета. В последнем столбце содержатся значения весовых коэффициентов, которые были присвоены цветовым образцам в результате работы предложенного алгоритма.



Также в таблице приведены значения максимальной  $\Delta E_{\text{макс}}$ , минимальной  $\Delta E_{\text{мин}}$  и средней  $\Delta E_{\text{ср}}$  ошибки для каждого метода.

За счет присвоения цветам флуоресценции наибольших весов, обеспечивается наилучшее соответствие цветовых координат опорного и калибруемого изображений. Цветам синей и желтой гаммы присваиваются промежуточные веса, поскольку именно эти цвета входят в канал *b*, на основе которого осуществляется классификация. Цвета, которые не являются критичными для процедуры классификации могут иметь большие рассогласования, поэтому, в данном случае, значения максимальной, средней и минимальной ошибок не могут быть использованы в качестве меры оценки работы алгоритма.

Для наглядности результатов работы ниже приведена гистограмма, отображающая величину ошибки цветопередачи для каждого цветового образца при использовании реализованных методов калибровки (рисунок 29).

Таблица 2

Значения ошибок воспроизведения цветов калибруемой камеры по отношению к опорной

| Образец<br>цвета  | $\Delta E_{\text{мнк}}$ | $\Delta E_{\text{итер.}}$ | $\Delta E_{\text{степень.знач.}}$<br>(предложенный<br>метод) | Весовые<br>коэффициент<br>ы |
|---|-------------------------|---------------------------|--|-----------------------------|
|    | 5,104519                | 0,000779                  | 0,025356   | 0,323972                    |
|    | 21,5727                 | 27,0297                   | 1,311266   | 0,286824                    |
|    | 2,642674                | 6,322923                  | 3,369345   | 0,103854                    |
|    | 5,294984                | 5,192236                  | 3,748274   | 0,082415                    |
|    | 8,072647                | 5,920951                  | 3,749134   | 0,039136                    |
|    | 2,815725                | 0,005845                  | 5,427144   | 0,037265                    |
|    | 1,724483                | 0,003712                  | 5,595082   | 0,034475                    |
|    | 3,28881                 | 0,004032                  | 2,140178   | 0,032701                    |
|   | 9,723735                | 6,76192                   | 6,402897   | 0,031614                    |
|  | 5,656057                | 0,002277                  | 11,71540   | 0,018010                    |
|  | 14,67964                | 19,05682                  | 27,79376   | 0,005164                    |
|  | 9,243423                | 12,0338                   | 18,07869   | 0,003421                    |
|  | 10,34274                | 11,8815                   | 15,74766   | 0,000551                    |
|  | 6,544934                | 7,848494                  | 9,813996   | 0,000440                    |
|  | 7,723348                | 8,756671                  | 9,246276   | 0,000158                    |
| $\Delta E_{\text{макс}}$  | 21,5727                 | 27,0297                   | 27,79376   |                             |
| $\Delta E_{\text{мин}}$   | 1,724483                | 0,000779                  | 0,025356   |                             |
| $\Delta E_{\text{ср}}$  | 7,628694                | 7,388195                  | 8,27763  |                             |

Введение матрицы весовых коэффициентов позволило обеспечить минимум ошибки на строго определенных цветах. Это достигается через присвоение цветам, для которых должны быть обеспечены сопоставимые количественные оценки, наибольших весовых коэффициентов. Так как, вектор весов был подвергнут сортировке, то наибольшие веса находятся вначале вектора. Как можно видеть на рисунке 29 при использовании предложенного метода минимум значений ошибок цветопередачи приходится на первые цветовые образцы.

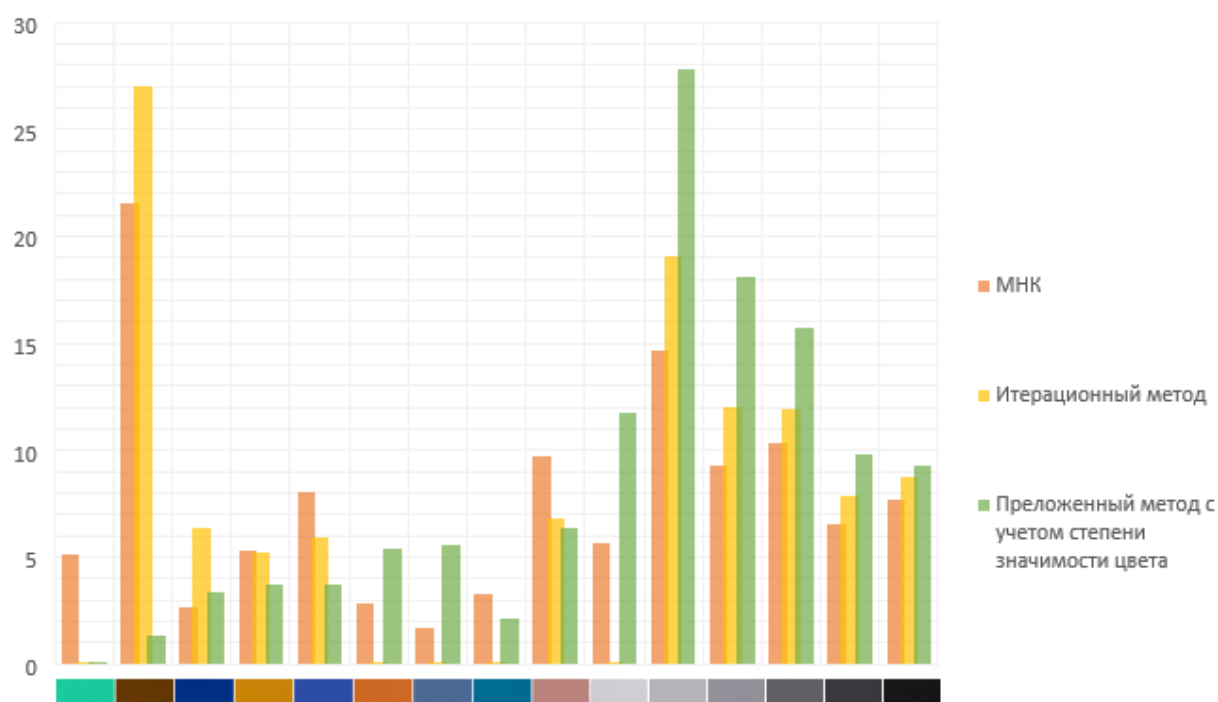


Рисунок 29 – Гистограмма значений ошибки цветопередачи для синтезированного набора цветов при использовании трех методов калибровки

Таким образом, по результатам исследований можно заключить, что предложенный метод с учетом степени значимости цветов обеспечивает минимум ошибки цветопередачи на определенных цветах, для которых необходимо обеспечить максимально достоверную передачу.

## 4. СПЕЦИАЛЬНЫЕ ВОПРОСЫ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ

### 4.1. Эргономический аспект разработки

Разработка интерактивных систем должна быть направлена на создание пригодных в использовании и полезных систем с учетом особенностей пользователей, их потребностей на основе эргономических принципов. Эргономический аспект разработки является одним из важнейших, поскольку обеспечивает соблюдения норм, отвечающих за благополучие человека при работе с системой, взаимодействие человека с ее элементами, оптимизации общей производительности системы. Такой подход к разработке систем называется человеко-ориентированным проектированием. Этот подход увеличивает результативность, эффективность, доступность и устойчивость систем, удовлетворенность пользователя и производительность его труда, а также предотвращает возможное неблагоприятное влияние использования систем на здоровье и безопасность человека. Принятие человеко-ориентированного подхода к проектированию и разработке несет существенную экономическую и социальную выгоду для пользователей, работодателей и поставщиков. Продукция и системы с высокой пригодностью использования имеют тенденцию быть более совершенными с технической точки зрения и коммерчески более успешными.

Системы, разработанные с использованием человеко-ориентированных методов, обладают более высоким качеством за счет:

- увеличения производительности пользователей и производительности труда в организациях;
- простоты понимания и использования, в результате чего снижается стоимость обучения и поддержки;
- учета опыта пользователей;
- снижения у пользователя дискомфорта и стресса;

Общая выгода принятия человеко-ориентированного принципа проектирования может быть определена с учетом стоимости всего жизненного цикла продукции, включая концепцию, проект, осуществление, поддержку, использование, техническое обслуживание и вывод из эксплуатации. Принятие человеко-ориентированного подхода к проектированию благотворно влияет на другие аспекты проектирования системы, например, упрощая идентификацию и формулирование функциональных требований.

Принятие человеко-ориентированного подхода также увеличивает вероятность успешного завершения проекта в срок и в рамках бюджета. Использование подходящих человеко-ориентированных методов может снизить риск несоответствия продукции требованиям причастных сторон и пользователей [21].

#### 4.2 Проектирование на основе точного определения пользователей, задач и среды

Продукция, системы и услуги должны быть разработаны таким образом, чтобы учитывать влияние, которое они могут оказать на все причастные стороны. Следовательно, все важные группы пользователей и причастных сторон должны быть определены. Построение систем на основе неверного или неполного понимания потребностей пользователей является одним из главных источников отказа системы. Степень пригодности и доступности зависит от условий использования, т. е. установленных пользователей, имеющих установленные цели, выполняющих установленные задачи в определенных условиях использования [1].

Программное обеспечение для калибровки медицинских камер рассчитано на использование медицинскими работниками, а именно гинекологами, которые проводят процедуру кольпоскопии, используя реализованную программу. Поскольку, коммерческие реализации медицинских систем обычно оснащаются камерами на КМОП-датчиках, которые имеют неидеальные характеристики чувствительности и не позволяют получить на выходе достоверное изображение. Корректная передача цветов необходимо для выполнения процедур классификации или сегментации медицинских изображений. Использование процедуры классификации позволяет определить вид заболевания (дифференциальная диагностика), находить синдромы, наиболее характерные для данного заболевания по совокупности симптомов или оценивать риск осложнений. В задаче сегментации реализуется разделение изображение на области интереса, например, имеющие одинаковую стадию патологии тканей. Поскольку данные процедуры осуществляют свою работу на основе цветового признака, то необходимо наличие сопоставимости его количественных оценок у изображений, полученных разными камерами или полученных в разных

условиях наблюдения. Выполнение этого условия определяет необходимость проведения специальной процедуры – цветовой калибровки.

#### 4.3. Условия использования

Характеристики пользователей, задач и вариантов среды называют условиями использования. Условия использования — это главный источник информации для установления требований пользователей и важный момент в процессе проектирования.

Разработанная процедура цветовой калибровки может применяться в различных прикладных телевизионных системах. К ним относятся космические, подводные, военные, медицинские и другие системы. Хотя процедура разрабатывалась применительно к медицинским ТВ системам, однако, главной особенностью является ручной выбор цветов, наиболее важных применительно к поставленной задаче. Для медицинских изображений этими цветами являются цвета синей и желтой гаммы, поскольку карта патологии строится на основе классификации флуоресцентных изображений по цветовым признакам b (система Lab) и Cr,Cb (система YCrCb). Однако, для других задач можно изменить набор цветовых образцов и осуществить калибровку с учетом интересующих цветов.

#### 4.4. Представления пользователя и разработчика о качестве программного обеспечения

Пользователи в основном проявляют заинтересованность в применении программного обеспечения, его производительности и результатах использования. Пользователи оценивают программное обеспечение без изучения его внутренних аспектов или того, как программное обеспечение создавалось.

Процесс создания требует от пользователя и разработчика использования одних и тех же характеристик качества программного обеспечения, так как они применяются для установления требований и приемки. Так как разработчики отвечают за создание программного обеспечения, которое должно удовлетворять требованиям качества, они заинтересованы в качестве промежуточной продукции так же, как и в качестве конечной продукции. Для того, чтобы оценить качество промежуточной продукции на каждой фазе

цикла разработки, разработчики должны использовать различные метрики для одних и тех же характеристик, потому что одни и те же метрики неприменимы для всех фаз жизненного цикла.

Вовлечение пользователей в проектирование и разработку является важным источником знаний об условиях использования, задачах, и о том, как пользователи будут работать с продуктом, системой или услугой. Вовлечение пользователя должно быть активным, он может участвовать в проектировании как источник важных данных или участвовать в оценке тех или иных решений. Способы и частота привлечения к проектированию пользователей изменяются в процессе проектирования и разработки и зависят от особенностей проекта. Результативность вовлечения пользователя возрастает с увеличением активности взаимодействия между разработчиками и пользователями [22].

#### 4.5. Функциональные требования к программному обеспечению

Описание функциональных возможностей и ограничений, накладываемых на программную систему, называется требованиями к этой системе, а сам процесс формирования, анализа, документирования и проверки этих функциональных возможностей и ограничений - разработкой требований (requirements engineering). В стандарте ГОСТ [23] рекомендуются следующие формулировки функциональных возможностей:

- Обзор функций

В описании продукта должен быть приведен обзор функций продукта, вызываемых пользователем, необходимых для них данных и предоставляемых средств.

- Выбор опорного изображения тестовой таблицы ColorChecker. Пользователю предоставляется выбор загружаемого в качестве опорного изображения тестовой таблицы ColorChecker. Изображение может иметь формат jpg, bmp, png.
- Вывод опорной тестовой таблицы в отдельное окно.
- Выбор калибруемого изображения тестовой таблицы ColorChecker. Пользователю предоставляется выбор загружаемого в качестве опорного изображения тестовой таблицы ColorChecker. Изображение может иметь формат jpg, bmp, png.
- Вывод калибруемой тестовой таблицы в отдельное окно.



- Выбор цветового образца. Считывание координат цветового образца производится путем нажатия левой кнопкой мыши по изображению тестовой таблицы.
- Расчёт ошибки цветопередачи между калибруемой и опорной тестовыми таблицами. Вывод значений ошибок производится в расположенные на форме таблицы.
- Расчет линейной матрицы преобразования. Вывод элементов матриц производится в расположенные на форме таблицы.
- Выбор калибруемого изображения. Пользователю предоставляется выбор изображения, к которому будет применена операция цветовой калибровки. Изображение может иметь формат jpg, bmp, png.
- Вывод оригинала калибруемого изображения и откалиброванного изображения на экран в отдельных окнах. Вывод изображений в отдельные окна позволяет пользователю манипулировать высотой и шириной окна, а также переносить его на дополнительные мониторы для удобства работы.

- Граничные значения

Если использование продукта ограничено конкретными граничными значениями для продукта, они должны быть указаны в описании продукта.

Минимальный объем выборки или набора цветовых образцов определяется:

- применялась ли предварительно к изображению процедура баланса белого;
- количеством цветов, необходимых для калибровки, в ходе решаемой задачи.

Если для изображения не требуется применение процедуры баланса белого, то количество цветов будет определяться только задачами и целями цветовой калибровки. Если для изображения не применялась процедура баланса белого, то минимальный набор цветов должен включать в себя 6 цветов градаций серого, расположенных на тестовой мишени, для выполнения этой процедуры.

#### 4.6. Интерфейс пользователя

С точки зрения пользователя операционная система формирует удобный пользовательский интерфейс, программное окружение, на фоне которого выполняется разработка и осуществляется исполнение прикладной программы пользователя.

Интерфейс пользователя — элементы и компоненты программы, которые способны оказывать влияние на взаимодействие пользователя с программным обеспечением. В том числе:

- средства отображения информации, отображаемая информация, форматы и коды;
- командные режимы, язык пользователь-интерфейс;
- устройства и технологии ввода данных;
- диалоги, взаимодействие и транзакции между пользователем и компьютером;
- обратная связь с пользователем;

В зависимости от типа пользовательского интерфейса информационные технологии имеют соответствующую классификацию. При этом выделяется системный и прикладной интерфейс. Прикладной интерфейс связан с реализацией некоторых функциональных информационных технологий. Системный интерфейс — это набор приемов взаимодействия с компьютером, который реализуется операционной системой или ее надстройкой.

Поскольку интерфейс является физическим динамическим устройством, взаимодействующим с пользователем, то наряду с абстрактно-синтаксическим возникает и дополняющий его независимый эргономический аспект, который, в зачаточной форме и соответствует обычному текстовому объекту (размер шрифта, цветовое оформление). В случае компьютерного интерфейса появляются новые особенности, связанные с комфортностью экранного представления, достаточной оперативностью реакции программного средства на действия пользователя, удобством манипулирования мышью и клавиатурой.

Нормативные требования по эргономике пользовательского интерфейса отличаются по своей природе от синтаксических и манипуляционных правил — они относятся к психофизиологическим свойствам конкретной реализации уже выбранного типа (стиля) пользовательского интерфейса (и соответствующего стандарта) в конкретном приложении. В этих условиях эргономические стандарты могут лишь требовать достижения некоторых

общих руководящих эргономических принципов [24], которым должно удовлетворять реализация в приложении выбранного типа. При этом предполагается, что приложение должно оптимально инкорпорировано в техническую среду. Ряд более ранних стандартов (стандарты ISO 9241 P.3-9) касаются именно этой среды (клавиатура, дисплеи, устройства ввода с клавиатуры и мыши, мебель рабочей станции и показатели рабочей среды, например, освещение или уровни шума). Эргономические аспекты пользовательского интерфейса приложения являются естественным расширением эргономики технических средств и рабочего места.

Сегодня существует два подхода к оценке эргономического качества, которые можно отнести к методам «черного» и «белого ящика».

В первом подходе оценку производит конечный пользователь, суммируя результаты работы с программой в рамках следующих показателей [25]:

- эффективности - влияния интерфейса на полноту и точность достижения пользователем целевых результатов;
- продуктивности или влияния интерфейса на производительность пользователя;
- степени (субъективной) удовлетворенности конечного пользователя этим интерфейсом.

Эффективность является критерием функциональности интерфейса, а степень удовлетворенности и, косвенно, продуктивность — критерием эргономичности.

Во втором подходе пытаются установить, каким (руководящим эргономическим) принципам должен удовлетворять пользовательский интерфейс с точки зрения оптимальности человеко-машинного взаимодействия. Развитие этого аналитического подхода было вызвано потребностями проектирования и разработки ПО, поскольку позволяет сформулировать руководящие указания по организации и характеристикам оптимального пользовательского интерфейса. Этот подход может быть использован и при оценке качества разработанного пользовательского интерфейса. В этом случае показатель качества оценивается экспертом по степени реализации руководящих принципов или вытекающих из них более конкретных графических и операционных особенностей оптимального «человеко-ориентированного» пользовательского интерфейса.

Программа реализована на языке объектно-ориентированного программирования C++ (Visual studio 2015) и имеет WIMP-интерфейс

(рисунок 30). При использовании WIMP-интерфейса (Windows Image Menu Pointer) на экране высвечивается окно, содержащее образы программ и меню действий. Для выбора одного из них используется указатель. В настоящее время практически все распространенные операционные системы предоставляют для своей работы графический интерфейс WIMP, использующий указательное устройство, выбор команд из меню, предоставление программам отдельных окон, использование для обозначения программ образов в виде пиктограмм. Удобство интерфейса и богатство возможностей делают Windows оптимальной системой для повседневной работы. Приложения, написанные под Windows, используют тот же интерфейс, поэтому его единообразие сводит к минимуму процесс обучения работе с любым приложением Windows.

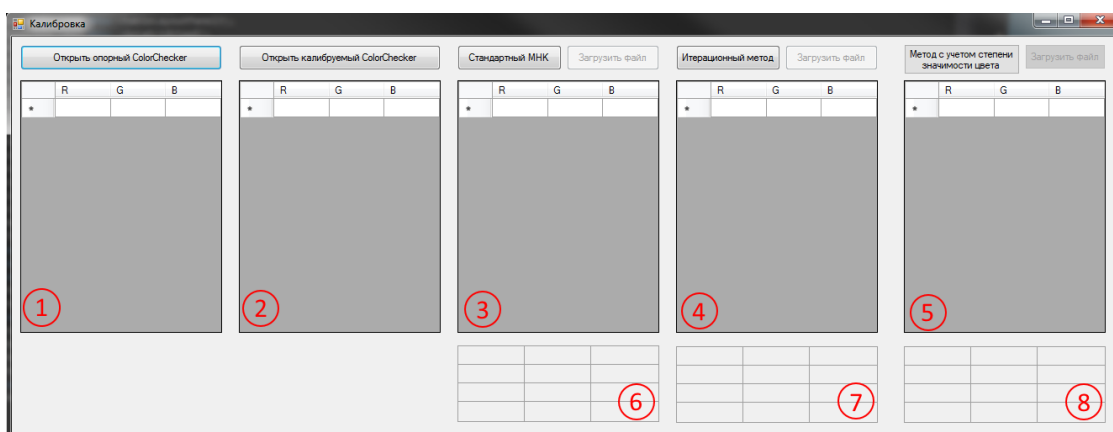


Рисунок 30 - Интерфейс программы

На форме программы расположены таблицы, содержащие:

- матрицы RGB-координат цветов опорного и калибруемого изображений (таблицы 1, 2)
- матрицы ошибок сведения цветов (таблицы 3, 4, 5)
- рассчитанные матрицы линейного преобразования (таблицы 6, 7, 8)

#### 4.7. Принципы организации диалога

##### 4.7.1. Приемлемость организации диалога для производственного задания

Принципы организации диалога применимы при разработке интерфейсов пользователя интерактивных систем и направлены на обеспечение пригодности использования этих систем для пользователей в части:

- выполнения действий, не предусмотренных производственным заданием;
- наличия в системе информации, вводящей пользователя в заблуждение;
- наличия недостаточной и неполной информации об интерфейсе пользователя;
- появления неожиданной ответной реакции интерактивной системы;
- ограничений в перемещении данных;
- в случае неэффективного восстановления работоспособности системы после ее отказа или сбоя.

Под диалогом понимается взаимодействие между пользователем и интерактивной системой, рассматриваемое как последовательность действий пользователя (ввод) и ответных реакций системы (вывод), направленное на обеспечение установленной цели. Стандарт ГОСТ [21] устанавливает семь основных принципов диалога, применяемых при проектировании и критериев оценки диалога:

- приемлемость организации диалога для выполнения производственного задания;
- информативность;
- соответствие ожиданиям пользователей;
- пригодность для обучения;
- контролируемость;
- устойчивость к ошибкам;
- адаптируемость к индивидуальным особенностям пользователей.

Диалог должен обеспечить пользователю успешное выполнение задачи с минимальными усилиями. Задачей реализованного программного продукта является осуществление операции цветовой калибровки, а именно нахождение оптимальной матрицы преобразования цветовых пространств, которая учитывает характеристики калибруемой камеры и осуществляет пересчет цветового пространства калибруемого изображения относительно опорного. Для выполнения этой задачи необходимо получить информацию о цветовых координатах каждого образца калибруемых цветов с помощью тестовых таблиц. С помощью дополнительных окон диалога, для удобства пользователя, на экран выводятся тестовые таблицы. Для оценки результатов работы алгоритма оригинальное и откалиброванное изображения также выводятся в отдельные окна.

#### 4.7.2. Информативность

Диалог должен быть информативен в такой степени, чтобы в любое время пользователю было ясно, в каком диалоге он находится и, если он находится в пределах диалога, какие действия и как могут быть выполнены. После того как пользователь произвел выбор цветовых образцов на опорной и калибруемой тестовых таблицах и, тем самым, произвел заполнение массивов RGB-координат, при нажатии на кнопки «Стандартный МНК», «Итерационный метод» и «Метод с учетом значимости цвета» производится расчёт матриц преобразования. Только после того как матрицы рассчитаны и их можно применить к калибруемым изображениям, будут доступны кнопки «Загрузить файл» для каждого метода (см. рисунок 30).

Если массивы RGB-координат опорной и калибруемой тестовых таблиц имеют неодинаковую размерность, то пользователь получит соответствующее уведомление (рисунок 31).

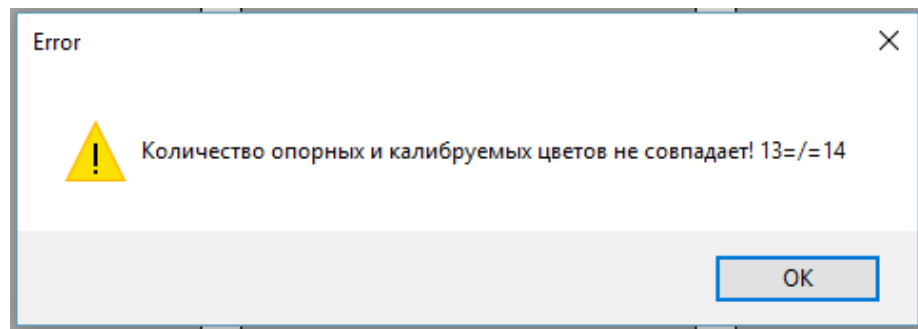


Рисунок 31 – Уведомления об ошибке

#### 4.7.3. Соответствие ожиданиям пользователей

Согласно стандарту, приложение должно соответствовать контекстуальным нуждам пользователей и общепринятым соглашениям. Контекст использования — это совокупность характеристик пользователей и их задач, а также организационной, технической и физической среды [23]. Для удобства пользователя интерфейс реализован на русском языке. Поскольку программа была рассчитана на русскоязычного медицинского работника, данный выбор является оптимальным. Терминология соответствует знаниям специалиста, обладающего компетенциями в области цифровой обработки изображений и знакомого с понятиями тестовых таблиц ColorChecker, используемых для калибровки цифровых камер. Пользователю необходимо четко понимать для каких целей осуществляется цветокалибровка. Если для целей сегментации и классификации - то по каким параметрам работают

применяемые алгоритмы, в какой цветовой системе и по каким признакам. Понимание и обладание знаниями о принципах работы всей конечной системы, позволит пользователю добиться наилучших результатов при работе с реализованным программным обеспечением.

Также для пользователя необходимо знание операционной системы, на которой будет установлена программа, и умение с ней работать. Основные функции программа выполняет автоматически, однако, пользователю будет необходимо самостоятельно выбирать необходимые файлы для обработки.

В ходе работы программы на форме одновременно отображаются матрицы ошибок сведения цветов и рассчитанные матрицы линейного преобразования (см. рисунок) для всех реализованных трёх методов. Для удобства пользователей можно реализовать выбор метода с помощью выпадающего списка. Основная цель выпадающего списка — отобразить список из некоторого числа элементов в ограниченной области экранного пространства и обеспечить возможность быстрого выбора. Также можно реализовать способ «drag-and-drop» на кнопках «Загрузить файл» - интерфейс переноса и приема компонентов, обеспечивающий взаимодействие двух элементов управления во время выполнения приложения. Однако, поскольку программа создавалась, в первую очередь, для научных целей и анализа результатов исследуемых методов, то одновременное отображение таблиц ошибок цветопередачи на форме программы является обязательным условием.

#### 4.7.4. Адаптируемость к индивидуальным особенностям пользователя

Диалог является адаптируемым к индивидуальным особенностям применения, если пользователи могут внести изменения в формат взаимодействия с системой и в формы представления информации для того, чтобы удовлетворить свои индивидуальные возможности и потребности [22].

Особенностью разработанной процедуры является самостоятельный выбор набора цветов для калибровки системы. В настоящее время сертифицированные программные продукты, позволяющие проводить калибровку цветовых пространств изображений и создающий цветовые профили на основе этих данных, имеют функцию автоматического определения местоположения тестовых таблиц ColorChecker на фотографиях, а также считывания цветовых координат всех образцов шкалы. Однако, для изображений для которых калибровка цветовых признаков является основой

для классификации и сегментации, должна быть возможность выбора именно тех цветов, для которых должна быть обеспечена максимально достоверная цветопередача. Поэтому, набор цветов для калибровки формируется пользователем самостоятельно, в зависимости от целей и задач дальнейшей работы с изображениями.

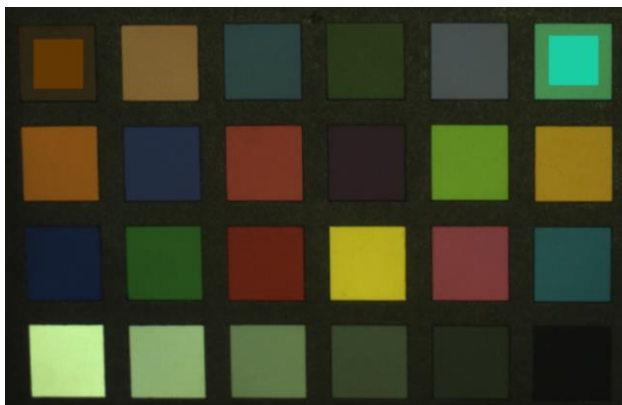


Рисунок 32 – Опорный ColorChecker

При нажатии на кнопку «Открыть опорный ColorChecker» на экран выводится опорное изображение, по которому выполняется калибровка (рисунок 32).

При нажатии на кнопку «Открыть калибруемый ColorChecker» на экран выводится калибруемое изображение (рисунок 33).

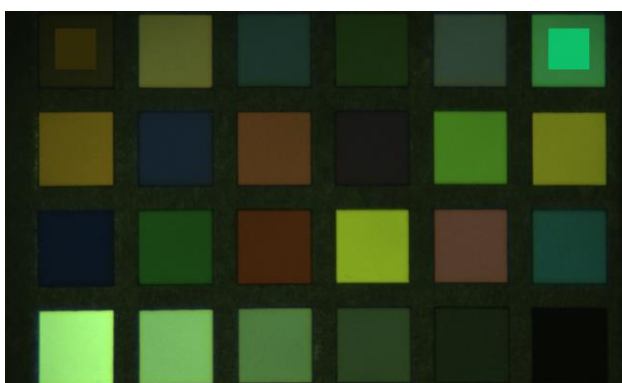


Рисунок 33 – Калибруемый ColorChecker

Пользователю предлагается сформировать набор тестовых образцов. Выбор цвета происходит нажатием кнопкой мыши на центр интересующего цветового образца.

В левом верхнем углу экрана располагаются два небольших окна для быстрого просмотра выбранных на тестовой таблице образцов цвета

(рисунок 34). Это дает возможность пользователю оценить визуальное рассогласование цветовых образцов.

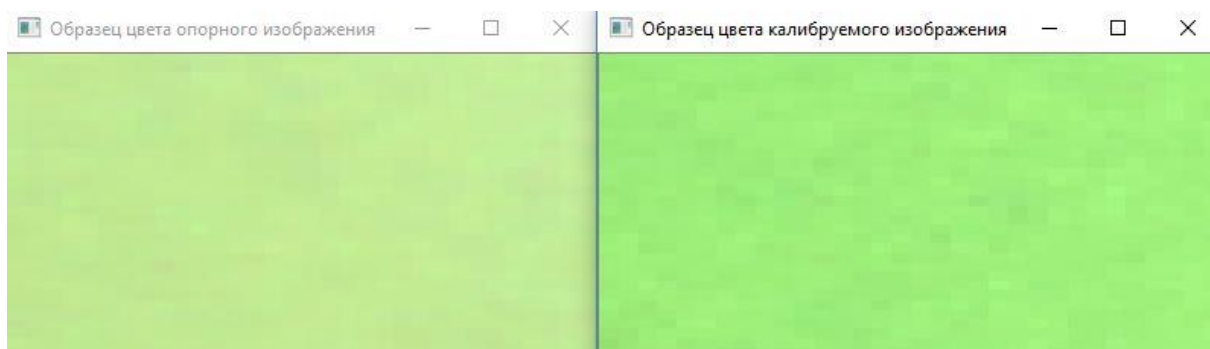


Рисунок 34 - Окна быстрого просмотра цветовых образцов



В процессе выбора пользователем цветовых образцов программа формирует два независимых массива цветовых координат опорного и калибруемого изображений. За счёт этого имеется возможность одновременно выбирать один и тот же образец на обоих изображениях.

После выбора изображения, к которому применяется процедура калибровки, программа автоматически осуществляет пересчёт цветового пространства выбранного изображения с учетом матрицы линейного преобразования. На форму будут выведены два окна, содержащие в себе оригинальное изображение и пересчитанное (рисунок 35).

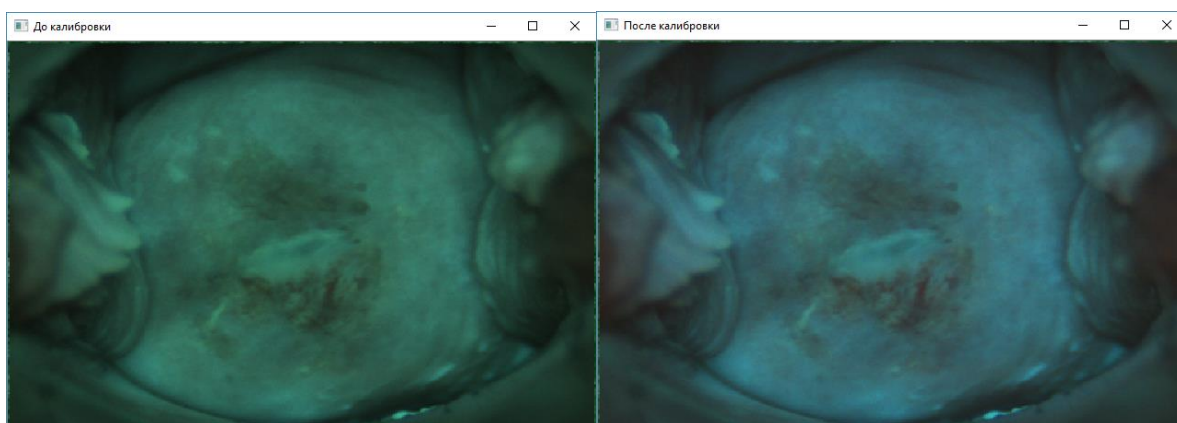


Рисунок 35 – Вид окон «До калибровки» и «После калибровки»

Вывод изображений в отдельные окна позволяет пользователю манипулировать высотой и шириной окна, а также переносить его на дополнительные мониторы для удобства работы.

## ЗАКЛЮЧЕНИЕ

В работе предложена процедура цветовой калибровки мультиспектральной системы медицинской диагностики. Особенностью данной процедуры является возможность синтеза нового тестового набора цветов и расчет матрицы преобразования цветовых пространств с использованием весовой функции.

Использование метода с весовой функцией позволило снизить ошибку цветопередачи в наиболее важных цветах. Данный метод учитывает степень значимости цветовых образцов с помощью присвоения им определенных весовых коэффициентов. В итерационном методе наименьших квадратов основной задачей является минимизация влияния выбросов, что, в результате, дает увеличение значения ошибки в других цветах, как правило, и в цветах необходимых для классификации. Предложенный метод минимизирует ошибки в конкретных цветах, поэтому увеличение рассогласования RGB-координат цветов, не участвующих в процедуре классификации, можно не учитывать.

Для проведения экспериментального исследования и оценки эффективности предложенной процедуры калибровки была разработана программа на языке объектно-ориентированного программирования C++ в среде Visual Studio 2015.

Предложенный метод с учетом значимости цвета имеет наибольшие значения максимальной  $\Delta E_{\text{макс}}$  и средней  $\Delta E_{\text{ср}}$  ошибок цветопередачи, однако, применительно к поставленной задаче, наиболее важным является то, что данный метод обеспечивает минимум ошибки на наиболее важных цветах. За счет присвоения цветам флюоресценции наибольших весов (цветовые образцы 1 и 2), обеспечивается наилучшее соответствие цветовых координат опорного и калибруемого изображений. В цветовом образце 1 значение ошибка цветопередачи по сравнению с методом наименьших квадратов уменьшилась на 5,079163. В цветовом образце 2 предложенный метод уменьшил ошибку на 20,5727 по сравнению с методом наименьших квадратов и на 25,718434 по сравнению с итерационным методом наименьших квадратов. Цвета, которые не являются критичными для процедуры классификации могут иметь большие рассогласования, поэтому, в данном случае, значения максимальной, средней и минимальной ошибок не могут быть использованы в качестве меры оценки работы алгоритма.

Дополнительная оценка эффективности предложенной процедуры была выполнена по критерию вероятности правильной классификации. Результат сравнения базы данных, полученной без применения цветокоррекции, с базой данных после цветокоррекции, показал, что предложенная процедура обеспечивает более эффективное восстановление информации о цветовой координате  $b$  и увеличивает вероятность правильной классификации на 20%. Этот результат можно объяснить тем, что при формировании тестового набора были использованы именно наиболее важные для задачи классификации цвета.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гонсалес Р., Вудс Р. Цифровая обработка изображений. М.: Техносфера, 2005. 560 с.
2. Многомерный анализ изображений в медицине и биологии / И.А. Беляев, С.В. Кучерявский, О.Е. Родионова, А.Л. Померанцев. // Энциклопедия инженера-химика. М.: Наука и Технологии, 2008. №2. С. 13-23.
3. Прэтт У. Цифровая обработка изображений. Т.01. М.: Мир, 1982. 312 с.
4. Фисенко В.Т., Фисенко Т.Ю. Компьютерная обработка и распознавание изображений: учеб. пособие. СПб: СПбГУ ИТМО, 2008. 7 с.
5. Грузман И.С., Киричук В.С., Косых В.П. и др. Цифровая обработка изображений в информационных системах: учеб. пособие. Новосибирск: Изд-во НГТУ, 2002. 247 с.
6. Обухова Н. А., Мотыко А. А. Калибровка по цвету в ТВ системах медицинской диагностики // «Цифровая обработка сигналов и ее применение DSPA - 2016»: тез. докл. 18 международной науч.-техн. конф., Москва, 30 март. - 01 апр. 2016 г. / РАН. Москва, 2016. С. 1-3.
7. Wolf S. Color Correction Matrix for Digital Still and Video Imaging Systems // URL: <https://www.its.bldrdoc.gov/publications/2437.aspx> (дата обращения: 28.05.2017)
8. Lam E. Y. Combining Gray World and RetinexTheory for Automatic White Balance in Digital Photography // URL: [https://www.eee.hku.hk/optima/pub/conference/0506\\_ISCE.pdf](https://www.eee.hku.hk/optima/pub/conference/0506_ISCE.pdf) (дата обращения: 28.05.2017)
9. Мишень X-Rite ColorChecker Grayscale // URL: <https://colorimetr.ru/21-x-rite-colorchecker-grayscale> (дата обращения: 20.05.2017).
10. Домасев М.В., Гнатюк С.П. Цвет, управление цветом, цветовые расчеты и измерения. СПб: Питер, 2009. 224 с.
11. Нормальное распределение (распределение Гаусса) // URL: <http://ee-system.ru/normalnoe-raspredelenie-raspredelenie-gaussa/> (дата обращения: 21.05.2017).
12. Свойства нормального распределения // URL: <http://smart-lab.ru/finansoviy-slovar/%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5%20%D1%80%D0%B0%D1%81%D0%BF%D1%80>

%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5/  
(дата обращения: 19.05.2017).

13. Вентцель Е.С. Теория вероятностей. М.: Высш. шк., 1999. 576 с.

14. Медведев А.В., Гринкевич А.В., Князева С.Н. Мультиспектральные системы различного назначения // Фотоника. 2015, вып. 05. С. 68-81.

15. Мультиспектральные флуоресцентные органоскопы для прижизненных исследований лабораторных животных и их органов / U. Kang, Г.В. Папаян, В.Б. Березин и др. // Оптический журнал. 2011, вып. 09. С. 82-90.

16. Комплекс для флуоресцентной диагностики и фотодинамической терапии заболеваний шейки матки / U. Kang, Г.В. Папаян, Н.А. Обухова и др. // Оптический журнал. 2015, вып. 12. С. 47-59.

17. Ferrari V., Tuytelaars T., Gool L.V. Simultaneous Object Recognition and Segmentation by Image Exploration // 8th European Conference on Computer Vision, May 11-14, 2004, Prague, Czech Republic. In Proc. ECCV. P. 40-54.

18. Русаков И.Г., Соколов В.В. Флуоресцентные методы диагностики и поверхностный рак мочевого пузыря: современное состояние проблемы // Урология. 2008, вып. 03. С. 67-71.

19. Флуоресцентная эндоскопия, дермаскопия и спектрофотометрия в диагностике злокачественных опухолей основных локализаций / В.И. Чиссов, В.В. Соколов, Н.Н. Булгакова и др. // Российский биотерапевтический журнал. 2003, вып. 04. С. 45-56.

20. Цветовые модели CMYK, RGB, Lab, HSB // URL: <http://ciframagazine.com/post.php?id=117> (дата обращения 15.05.2017)

21. ГОСТ Р ИСО 9241-210-2016. Описание стандартов. М.: Стандартинформ, 2016.

22. ГОСТ Р ИСО/МЭК 9126-93. Описание стандартов. М.: Госстандарт России, 1993.

23. ГОСТ Р ИСО/МЭК 12119-2000. Описание стандартов. М.: Госстандарт России, 2000.

24. Мандел Т. Разработка пользовательского интерфейса. М.: ДМК Пресс, 2001

25. ISO 9241-10-98 // URL: <http://www.userfocus.co.uk/pdf/ISO9241.pdf> (дата обращения 29.05.2017)

## ПРИЛОЖЕНИЕ

## Реализация глобальных методов контрастирования на языке программирования C++ с использованием библиотек OpenCV

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <cmath>
#include <msclr\marshal_cppstd.h>
#include <random>
#include <map>
#include <math.h>

#pragma once

namespace Calibration {
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    int m_0 = 0, m_P = 0; //Количество опорных цветов
    int roi_size = 32; //Размер окна для усреднения цвета
    cv::Mat colorChecker_0, colorChecker_P; //Изображения
    колорчекеров
    double **O, **P; //Массивы, содержащие цветовые координаты
    опорных цветов оригинального и калибруемого колорчекера
    double **M12, **M12_mod, **M12_w; //Матрицы линейного
    преобразования цветов без минимизации и с минимизацией выбросов
    double err = 0.0001; //Требуемая точность матрицы линейного
    преобразования
    double m_dist = 1;
    double s_dist = 0.3;

    void myMouseCallback(int event, int x, int y, int flags, void*
    param);
    void myMouseCallback2(int event, int x, int y, int flags, void*
    param);
    double **AddColumn(double **, int);
    int gaus_obr(int cnt_str, double **mass, double **&M_obr);
    double ** Transp(double **, int, int);
}
```

```

        double ** Multiplication(double **matrix_1, double **matrix_2,
int m, int n, int p); //Перемножение матриц
        double ** Difference(double **matrix_1, double **matrix_2, int m,
int n); //Вычитание матриц
        double * normal(const double m, const double s);

public ref class MyForm : public System::Windows::Forms::Form
{
public:
    MyForm(void)
    {
        InitializeComponent();
    }

protected:
    ~MyForm()
    {
        if (components)
        {
            delete components;
        }
    }

private:
    System::Windows::Forms::Button^ button1;
    private: System::Windows::Forms::Button^ button2;
    public: System::Windows::Forms::DataGridView^ dataGridView1;
    private: System::Windows::Forms::DataGridViewTextBoxColumn^ R;
    private: System::Windows::Forms::DataGridViewTextBoxColumn^ G;
    private: System::Windows::Forms::DataGridViewTextBoxColumn^ B;
    public: System::Windows::Forms::DataGridView^ dataGridView2;
    private:
    private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn1;
    public:
    private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn2;
    private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn3;
    private: System::Windows::Forms::Button^ button3;
    private: System::Windows::Forms::Button^ button4;
    public: System::Windows::Forms::DataGridView^ dataGridView3;
    private:
    private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn4;
    public:
    private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn5;
    private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn6;

```



```

        private: System::Windows::Forms::TableLayoutPanel^
tableLayoutPanel1;
        private: System::Windows::Forms::Label^ label3;
        private: System::Windows::Forms::Label^ label1;
        private: System::Windows::Forms::Label^ label4;
        private: System::Windows::Forms::Label^ label2;
        private: System::Windows::Forms::Label^ label12;
        private: System::Windows::Forms::Label^ label11;
        private: System::Windows::Forms::Label^ label5;
        private: System::Windows::Forms::Label^ label6;
        private: System::Windows::Forms::Label^ label7;
        private: System::Windows::Forms::Label^ label8;
        private: System::Windows::Forms::Label^ label9;
        private: System::Windows::Forms::Label^ label10;
        public: System::Windows::Forms::DataGridView^ dataGridView4;
        private:
        private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn7;
        public:
        private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn8;
        private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn9;
        private: System::Windows::Forms::TableLayoutPanel^
tableLayoutPanel2;
        private: System::Windows::Forms::Label^ label13;
        private: System::Windows::Forms::Label^ label15;
        private: System::Windows::Forms::Label^ label14;
        private: System::Windows::Forms::Label^ label16;
        private: System::Windows::Forms::Label^ label17;
        private: System::Windows::Forms::Label^ label18;
        private: System::Windows::Forms::Label^ label19;
        private: System::Windows::Forms::Label^ label20;
        private: System::Windows::Forms::Label^ label21;
        private: System::Windows::Forms::Label^ label22;
        private: System::Windows::Forms::Label^ label24;
        private: System::Windows::Forms::Label^ label23;
        private: System::Windows::Forms::Button^ button5;
        private: System::Windows::Forms::OpenFileDialog^
openFileDialog1;
        private: System::Windows::Forms::Button^ button6;
        private: System::Windows::Forms::Button^ button7;
        private: System::Windows::Forms::TableLayoutPanel^
tableLayoutPanel3;
        private: System::Windows::Forms::Label^ label25;
        private: System::Windows::Forms::Label^ label26;
        private: System::Windows::Forms::Label^ label27;
        private: System::Windows::Forms::Label^ label28;
        private: System::Windows::Forms::Label^ label29;
        private: System::Windows::Forms::Label^ label30;
        private: System::Windows::Forms::Label^ label31;

```

```

        private: System::Windows::Forms::Label^ label132;
        private: System::Windows::Forms::Label^ label133;
        private: System::Windows::Forms::Label^ label134;
        private: System::Windows::Forms::Label^ label135;
        private: System::Windows::Forms::Label^ label136;
        public: System::Windows::Forms::DataGridView^ dataGridView5;
        private:
        private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn10;
        public:
        private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn11;
        private: System::Windows::Forms::DataGridViewTextBoxColumn^
dataGridViewTextBoxColumn12;
        private: System::Windows::Forms::Button^ button8;
private:

```

```

        System::ComponentModel::Container ^components;

```

```

#pragma region Windows Form Designer generated code

```

```

        void InitializeComponent(void)
        {
            this->button1 = (gcnew
System::Windows::Forms::Button());
            this->button2 = (gcnew
System::Windows::Forms::Button());
            this->dataGridView1 = (gcnew
System::Windows::Forms::DataGridView());
            this->R = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
            this->G = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
            this->B = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
            this->dataGridView2 = (gcnew
System::Windows::Forms::DataGridView());
            this->dataGridViewTextBoxColumn1 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
            this->dataGridViewTextBoxColumn2 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
            this->dataGridViewTextBoxColumn3 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
            this->button3 = (gcnew
System::Windows::Forms::Button());
            this->button4 = (gcnew
System::Windows::Forms::Button());
            this->dataGridView3 = (gcnew
System::Windows::Forms::DataGridView());
            this->dataGridViewTextBoxColumn4 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());

```

```

        this->dataGridViewTextBoxColumn5 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
        this->dataGridViewTextBoxColumn6 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
        this->tableLayoutPanel1 = (gcnew
System::Windows::Forms::TableLayoutPanel());
        this->label1 = (gcnew
System::Windows::Forms::Label());
        this->label2 = (gcnew
System::Windows::Forms::Label());
        this->label3 = (gcnew
System::Windows::Forms::Label());
        this->label4 = (gcnew
System::Windows::Forms::Label());
        this->label5 = (gcnew
System::Windows::Forms::Label());
        this->label6 = (gcnew
System::Windows::Forms::Label());
        this->label7 = (gcnew
System::Windows::Forms::Label());
        this->label8 = (gcnew
System::Windows::Forms::Label());
        this->label9 = (gcnew
System::Windows::Forms::Label());
        this->label10 = (gcnew
System::Windows::Forms::Label());
        this->label11 = (gcnew
System::Windows::Forms::Label());
        this->dataGridView4 = (gcnew
System::Windows::Forms::DataGridView());
        this->dataGridViewTextBoxColumn7 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
        this->dataGridViewTextBoxColumn8 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
        this->dataGridViewTextBoxColumn9 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
        this->tableLayoutPanel2 = (gcnew
System::Windows::Forms::TableLayoutPanel());
        this->label13 = (gcnew
System::Windows::Forms::Label());
        this->label15 = (gcnew
System::Windows::Forms::Label());
        this->label14 = (gcnew
System::Windows::Forms::Label());
        this->label16 = (gcnew
System::Windows::Forms::Label());
        this->label17 = (gcnew
System::Windows::Forms::Label());

```

```

        this->label118 = (gcnew
System::Windows::Forms::Label());
        this->label119 = (gcnew
System::Windows::Forms::Label());
        this->label120 = (gcnew
System::Windows::Forms::Label());
        this->label121 = (gcnew
System::Windows::Forms::Label());
        this->label122 = (gcnew
System::Windows::Forms::Label());
        this->label124 = (gcnew
System::Windows::Forms::Label());
        this->label123 = (gcnew
System::Windows::Forms::Label());
        this->button5 = (gcnew
System::Windows::Forms::Button());
        this->openFileDialog1 = (gcnew
System::Windows::Forms::OpenFileDialog());
        this->button6 = (gcnew
System::Windows::Forms::Button());
        this->button7 = (gcnew
System::Windows::Forms::Button());
        this->tableLayoutPanel3 = (gcnew
System::Windows::Forms::TableLayoutPanel());
        this->label125 = (gcnew
System::Windows::Forms::Label());
        this->label128 = (gcnew
System::Windows::Forms::Label());
        this->label129 = (gcnew
System::Windows::Forms::Label());
        this->label130 = (gcnew
System::Windows::Forms::Label());
        this->label131 = (gcnew
System::Windows::Forms::Label());
        this->label132 = (gcnew
System::Windows::Forms::Label());
        this->label133 = (gcnew
System::Windows::Forms::Label());
        this->label134 = (gcnew
System::Windows::Forms::Label());
        this->label136 = (gcnew
System::Windows::Forms::Label());
        this->label135 = (gcnew
System::Windows::Forms::Label());
        this->label127 = (gcnew
System::Windows::Forms::Label());
        this->label126 = (gcnew
System::Windows::Forms::Label());
        this->dataGridView5 = (gcnew
System::Windows::Forms::DataGridView());

```

```

        this->dataGridViewTextBoxColumn10 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
        this->dataGridViewTextBoxColumn11 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
        this->dataGridViewTextBoxColumn12 = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
        this->button8 = (gcnew
System::Windows::Forms::Button());

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
->dataGridView1))->BeginInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
->dataGridView2))->BeginInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
->dataGridView3))->BeginInit();
        this->tableLayoutPanel1->SuspendLayout();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
->dataGridView4))->BeginInit();
        this->tableLayoutPanel2->SuspendLayout();
        this->tableLayoutPanel3->SuspendLayout();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
->dataGridView5))->BeginInit();
        this->SuspendLayout();
//
        // button1
        //
        this->button1->Location = System::Drawing::Point(10,
10);

        this->button1->Name = L"button1";
        this->button1->Size = System::Drawing::Size(240, 30);
        this->button1->TabIndex = 2;
        this->button1->Text = L"Открыть опорный ColorChecker";
        this->button1->UseVisualStyleBackColor = true;
        this->button1->Click += gcnew
System::EventHandler(this, &MyForm::button1_Click);
        //
        // button2
        //
        this->button2->Location = System::Drawing::Point(270,
10);

        this->button2->Name = L"button2";
        this->button2->Size = System::Drawing::Size(240, 30);
        this->button2->TabIndex = 3;
        this->button2->Text = L"Открыть калибруемый
ColorChecker";
        this->button2->UseVisualStyleBackColor = true;

```

```

        this->button2->Click += gcnew
System::EventHandler(this, &MyForm::button2_Click);
        //
        // dataGridView1
        //
        this->dataGridView1->AutoSizeColumnsMode =
System::Windows::Forms::DataGridViewAutoSizeColumnsMode::Fill;
        this->dataGridView1->ColumnHeadersHeightSizeMode =
System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoS
ize;
        this->dataGridView1->Columns->AddRange(gcnew
cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {
            this->R, this->G, this->B
        });
        this->dataGridView1->Location =
System::Drawing::Point(10, 52);
        this->dataGridView1->Name = L"dataGridView1";
        this->dataGridView1->RowTemplate->Height = 24;
        this->dataGridView1->Size = System::Drawing::Size(240,
300);

        this->dataGridView1->TabIndex = 4;
        //
        // R
        //
        this->R->HeaderText = L"R";
        this->R->Name = L"R";
        this->R->ReadOnly = true;
        //
        // G
        //
        this->G->HeaderText = L"G";
        this->G->Name = L"G";
        this->G->ReadOnly = true;
        //
        // B
        //
        this->B->HeaderText = L"B";
        this->B->Name = L"B";
        this->B->ReadOnly = true;
        //
        // dataGridView2
        //
        this->dataGridView2->AutoSizeColumnsMode =
System::Windows::Forms::DataGridViewAutoSizeColumnsMode::Fill;
        this->dataGridView2->ColumnHeadersHeightSizeMode =
System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoS
ize;
        this->dataGridView2->Columns->AddRange(gcnew
cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {
            this->dataGridViewTextBoxColumn1,

```

```

        this->dataGridViewTextBoxColumn2, this->dataGridViewTextBoxColumn3
    });
    this->dataGridView2->Location =
System::Drawing::Point(270, 52);
    this->dataGridView2->Name = L"dataGridView2";
    this->dataGridView2->RowTemplate->Height = 24;
    this->dataGridView2->Size = System::Drawing::Size(240,
300);

    this->dataGridView2->TabIndex = 5;
    //
    // dataGridViewTextBoxColumn1
    //
    this->dataGridViewTextBoxColumn1->HeaderText = L"R";
    this->dataGridViewTextBoxColumn1->Name =
L"dataGridViewTextBoxColumn1";
    this->dataGridViewTextBoxColumn1->ReadOnly = true;
    //
    // dataGridViewTextBoxColumn2
    //
    this->dataGridViewTextBoxColumn2->HeaderText = L"G";
    this->dataGridViewTextBoxColumn2->Name =
L"dataGridViewTextBoxColumn2";
    this->dataGridViewTextBoxColumn2->ReadOnly = true;
    //
    // dataGridViewTextBoxColumn3
    //
    this->dataGridViewTextBoxColumn3->HeaderText = L"B";
    this->dataGridViewTextBoxColumn3->Name =
L"dataGridViewTextBoxColumn3";
    this->dataGridViewTextBoxColumn3->ReadOnly = true;
    //
    // button3
    //
    this->button3->Location = System::Drawing::Point(530,
10);

    this->button3->Name = L"button3";
    this->button3->Size = System::Drawing::Size(124, 30);
    this->button3->TabIndex = 6;
    this->button3->Text = L"Стандартный МНК";
    this->button3->UseVisualStyleBackColor = true;
    this->button3->Click += gcnew
System::EventHandler(this, &MyForm::button3_Click);
    //
    // button4
    //
    this->button4->Location = System::Drawing::Point(790,
10);

    this->button4->Name = L"button4";
    this->button4->Size = System::Drawing::Size(124, 30);
    this->button4->TabIndex = 7;

```



```

        this->button4->Text = L"Итерационный метод";
        this->button4->UseVisualStyleBackColor = true;
        this->button4->Click += gcnew
System::EventHandler(this, &MyForm::button4_Click);
        //
        // dataGridView3
        //
        this->dataGridView3->AutoSizeColumnsMode =
System::Windows::Forms::DataGridViewAutoSizeColumnsMode::Fill;
        this->dataGridView3->ColumnHeadersHeightSizeMode =
System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoS
ize;
        this->dataGridView3->Columns->AddRange(gcnew
cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {
            this->dataGridViewTextBoxColumn4,
            this->dataGridViewTextBoxColumn5, this-
>dataGridViewTextBoxColumn6
        });
        this->dataGridView3->Location =
System::Drawing::Point(530, 52);
        this->dataGridView3->Name = L"dataGridView3";
        this->dataGridView3->RowTemplate->Height = 24;
        this->dataGridView3->Size = System::Drawing::Size(240,
300);
        this->dataGridView3->TabIndex = 8;
        this->dataGridView3->CellContentClick += gcnew
System::Windows::Forms::DataGridViewCellEventHandler(this,
&MyForm::dataGridView3_CellContentClick);
        //
        // dataGridViewTextBoxColumn4
        //
        this->dataGridViewTextBoxColumn4->HeaderText = L"R";
        this->dataGridViewTextBoxColumn4->Name =
L"dataGridViewTextBoxColumn4";
        this->dataGridViewTextBoxColumn4->ReadOnly = true;
        //
        // dataGridViewTextBoxColumn5
        //
        this->dataGridViewTextBoxColumn5->HeaderText = L"G";
        this->dataGridViewTextBoxColumn5->Name =
L"dataGridViewTextBoxColumn5";
        this->dataGridViewTextBoxColumn5->ReadOnly = true;
        //
        // dataGridViewTextBoxColumn6
        //
        this->dataGridViewTextBoxColumn6->HeaderText = L"B";
        this->dataGridViewTextBoxColumn6->Name =
L"dataGridViewTextBoxColumn6";
        this->dataGridViewTextBoxColumn6->ReadOnly = true;
        //
        // tableLayoutPanel1

```



```

//
this->tableLayoutPanel1->BackColor =
System::Drawing::SystemColors::Control;
this->tableLayoutPanel1->CellStyle =
System::Windows::Forms::TableLayoutPanelCellStyle::Single;
this->tableLayoutPanel1->ColumnCount = 3;
this->tableLayoutPanel1->ColumnStyles->Add((gcnew
System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::
Percent,
33.33F)));
this->tableLayoutPanel1->ColumnStyles->Add((gcnew
System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::
Percent,
33.33F)));
this->tableLayoutPanel1->ColumnStyles->Add((gcnew
System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::
Percent,
33.34F)));
this->tableLayoutPanel1->Controls->Add(this->label1,
0, 0);
this->tableLayoutPanel1->Controls->Add(this->label2,
1, 0);
this->tableLayoutPanel1->Controls->Add(this->label3,
2, 0);
this->tableLayoutPanel1->Controls->Add(this->label4,
0, 1);
this->tableLayoutPanel1->Controls->Add(this->label5,
1, 1);
this->tableLayoutPanel1->Controls->Add(this->label6,
2, 1);
this->tableLayoutPanel1->Controls->Add(this->label7,
0, 2);
this->tableLayoutPanel1->Controls->Add(this->label8,
1, 2);
this->tableLayoutPanel1->Controls->Add(this->label9,
2, 2);
this->tableLayoutPanel1->Controls->Add(this->label10,
0, 3);
this->tableLayoutPanel1->Controls->Add(this->label12,
2, 3);
this->tableLayoutPanel1->Controls->Add(this->label11,
1, 3);
this->tableLayoutPanel1->Location =
System::Drawing::Point(530, 367);
this->tableLayoutPanel1->Name = L"tableLayoutPanel1";
this->tableLayoutPanel1->RowCount = 4;
this->tableLayoutPanel1->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));

```

```

        this->tableLayoutPanel1->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel1->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel1->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel1->Size =
System::Drawing::Size(240, 91);
        this->tableLayoutPanel1->TabIndex = 9;
        //
        // label1
        //
        this->label1->AutoSize = true;
        this->label1->Location = System::Drawing::Point(4, 1);
        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(35, 13);
        this->label1->TabIndex = 0;
        this->label1->Text = L"label1";
        this->label1->Visible = false;
        //
        // label2
        //
        this->label2->AutoSize = true;
        this->label2->Location = System::Drawing::Point(83,
1);

        this->label2->Name = L"label2";
        this->label2->Size = System::Drawing::Size(35, 13);
        this->label2->TabIndex = 7;
        this->label2->Text = L"label2";
        this->label2->Visible = false;
        //
        // label3
        //
        this->label3->AutoSize = true;
        this->label3->Location = System::Drawing::Point(162,
1);

        this->label3->Name = L"label3";
        this->label3->Size = System::Drawing::Size(35, 13);
        this->label3->TabIndex = 8;
        this->label3->Text = L"label3";
        this->label3->Visible = false;
        //
        // label4
        //
        this->label4->AutoSize = true;
        this->label4->Location = System::Drawing::Point(4,
23);

        this->label4->Name = L"label4";

```

```

this->label4->Size = System::Drawing::Size(35, 13);
this->label4->TabIndex = 9;
this->label4->Text = L"label4";
this->label4->Visible = false;
//
// label5
//
this->label5->AutoSize = true;
this->label5->Location = System::Drawing::Point(83,
23);

this->label5->Name = L"label5";
this->label5->Size = System::Drawing::Size(35, 13);
this->label5->TabIndex = 10;
this->label5->Text = L"label5";
this->label5->Visible = false;
//
// label6
//
this->label6->AutoSize = true;
this->label6->Location = System::Drawing::Point(162,
23);

this->label6->Name = L"label6";
this->label6->Size = System::Drawing::Size(35, 13);
this->label6->TabIndex = 11;
this->label6->Text = L"label6";
this->label6->Visible = false;
//
// label7
//
this->label7->AutoSize = true;
this->label7->Location = System::Drawing::Point(4,
45);

this->label7->Name = L"label7";
this->label7->Size = System::Drawing::Size(35, 13);
this->label7->TabIndex = 12;
this->label7->Text = L"label7";
this->label7->Visible = false;
//
// label8
//
this->label8->AutoSize = true;
this->label8->Location = System::Drawing::Point(83,
45);

this->label8->Name = L"label8";
this->label8->Size = System::Drawing::Size(35, 13);
this->label8->TabIndex = 13;
this->label8->Text = L"label8";
this->label8->Visible = false;
//
// label9
//

```

```

        this->label9->AutoSize = true;
        this->label9->Location = System::Drawing::Point(162,
45);

        this->label9->Name = L"label9";
        this->label9->Size = System::Drawing::Size(35, 13);
        this->label9->TabIndex = 14;
        this->label9->Text = L"label9";
        this->label9->Visible = false;

//
// label10
//
        this->label10->AutoSize = true;
        this->label10->Location = System::Drawing::Point(4,
67);

        this->label10->Name = L"label10";
        this->label10->Size = System::Drawing::Size(41, 13);
        this->label10->TabIndex = 15;
        this->label10->Text = L"label10";
        this->label10->Visible = false;
//
// label12
//
        this->label12->AutoSize = true;
        this->label12->Location = System::Drawing::Point(162,
67);

        this->label12->Name = L"label12";
        this->label12->Size = System::Drawing::Size(41, 13);
        this->label12->TabIndex = 17;
        this->label12->Text = L"label12";
        this->label12->Visible = false;
//
// label11
//
        this->label11->AutoSize = true;
        this->label11->Location = System::Drawing::Point(83,
67);

        this->label11->Name = L"label11";
        this->label11->Size = System::Drawing::Size(41, 13);
        this->label11->TabIndex = 16;
        this->label11->Text = L"label11";
        this->label11->Visible = false;
//
// dataGridView4
//
        this->dataGridView4->AutoSizeColumnsMode =
System::Windows::Forms::DataGridViewAutoSizeColumnsMode::Fill;
        this->dataGridView4->ColumnHeadersHeightSizeMode =
System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoS
ize;
        this->dataGridView4->Columns->AddRange(gcnew
cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

```

```

        this->dataGridViewTextBoxColumn7,
        this->dataGridViewTextBoxColumn8, this->dataGridViewTextBoxColumn9
    });
    this->dataGridView4->Location =
System::Drawing::Point(790, 52);
    this->dataGridView4->Name = L"dataGridView4";
    this->dataGridView4->RowTemplate->Height = 24;
    this->dataGridView4->Size = System::Drawing::Size(240,
300);
    this->dataGridView4->TabIndex = 10;
    //
    // dataGridViewTextBoxColumn7
    //
    this->dataGridViewTextBoxColumn7->HeaderText = L"R";
    this->dataGridViewTextBoxColumn7->Name =
L"dataGridViewTextBoxColumn7";
    this->dataGridViewTextBoxColumn7->ReadOnly = true;
    //
    // dataGridViewTextBoxColumn8
    //
    this->dataGridViewTextBoxColumn8->HeaderText = L"G";
    this->dataGridViewTextBoxColumn8->Name =
L"dataGridViewTextBoxColumn8";
    this->dataGridViewTextBoxColumn8->ReadOnly = true;
    //
    // dataGridViewTextBoxColumn9
    //
    this->dataGridViewTextBoxColumn9->HeaderText = L"B";
    this->dataGridViewTextBoxColumn9->Name =
L"dataGridViewTextBoxColumn9";
    this->dataGridViewTextBoxColumn9->ReadOnly = true;
    //
    // tableLayoutPanel12
    //
    this->tableLayoutPanel12->BackColor =
System::Drawing::SystemColors::Control;
    this->tableLayoutPanel12->CellBorderStyle =
System::Windows::Forms::TableLayoutPanelCellBorderStyle::Single;
    this->tableLayoutPanel12->ColumnCount = 3;
    this->tableLayoutPanel12->ColumnStyles->Add((gcnew
System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::
Percent,
        33.33F)));
    this->tableLayoutPanel12->ColumnStyles->Add((gcnew
System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::
Percent,
        33.33F)));
    this->tableLayoutPanel12->ColumnStyles->Add((gcnew
System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::
Percent,

```

```

        33.34F));
        this->tableLayoutPanel2->Controls->Add(this->label13,
0, 0);
        this->tableLayoutPanel2->Controls->Add(this->label15,
2, 0);
        this->tableLayoutPanel2->Controls->Add(this->label14,
1, 0);
        this->tableLayoutPanel2->Controls->Add(this->label16,
0, 1);
        this->tableLayoutPanel2->Controls->Add(this->label17,
1, 1);
        this->tableLayoutPanel2->Controls->Add(this->label18,
2, 1);
        this->tableLayoutPanel2->Controls->Add(this->label19,
0, 2);
        this->tableLayoutPanel2->Controls->Add(this->label20,
1, 2);
        this->tableLayoutPanel2->Controls->Add(this->label21,
2, 2);
        this->tableLayoutPanel2->Controls->Add(this->label22,
0, 3);
        this->tableLayoutPanel2->Controls->Add(this->label24,
2, 3);
        this->tableLayoutPanel2->Controls->Add(this->label23,
1, 3);
        this->tableLayoutPanel2->Location =
System::Drawing::Point(790, 368);
        this->tableLayoutPanel2->Name = L"tableLayoutPanel2";
        this->tableLayoutPanel2->RowCount = 4;
        this->tableLayoutPanel2->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel2->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel2->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel2->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel2->Size =
System::Drawing::Size(240, 91);
        this->tableLayoutPanel2->TabIndex = 11;
        //
        // label13
        //
        this->label13->AutoSize = true;
        this->label13->Location = System::Drawing::Point(4,
1);
        this->label13->Name = L"label13";

```

```

this->label13->Size = System::Drawing::Size(41, 13);
this->label13->TabIndex = 17;
this->label13->Text = L"label13";
this->label13->Visible = false;
//
// label15
//
this->label15->AutoSize = true;
this->label15->Location = System::Drawing::Point(162,
1);

this->label15->Name = L"label15";
this->label15->Size = System::Drawing::Size(41, 13);
this->label15->TabIndex = 0;
this->label15->Text = L"label15";
this->label15->Visible = false;
//
// label14
//
this->label14->AutoSize = true;
this->label14->Location = System::Drawing::Point(83,
1);

this->label14->Name = L"label14";
this->label14->Size = System::Drawing::Size(41, 13);
this->label14->TabIndex = 16;
this->label14->Text = L"label14";
this->label14->Visible = false;
//
// label16
//
this->label16->AutoSize = true;
this->label16->Location = System::Drawing::Point(4,
23);

this->label16->Name = L"label16";
this->label16->Size = System::Drawing::Size(41, 13);
this->label16->TabIndex = 7;
this->label16->Text = L"label16";
this->label16->Visible = false;
//
// label17
//
this->label17->AutoSize = true;
this->label17->Location = System::Drawing::Point(83,
23);

this->label17->Name = L"label17";
this->label17->Size = System::Drawing::Size(41, 13);
this->label17->TabIndex = 8;
this->label17->Text = L"label17";
this->label17->Visible = false;
//
// label18
//

```

```

23);
    this->label18->AutoSize = true;
    this->label18->Location = System::Drawing::Point(162,

    this->label18->Name = L"label18";
    this->label18->Size = System::Drawing::Size(41, 13);
    this->label18->TabIndex = 9;
    this->label18->Text = L"label18";
    this->label18->Visible = false;
    //
    // label19
    //
    this->label19->AutoSize = true;
    this->label19->Location = System::Drawing::Point(4,

45);

    this->label19->Name = L"label19";
    this->label19->Size = System::Drawing::Size(41, 13);
    this->label19->TabIndex = 10;
    this->label19->Text = L"label19";
    this->label19->Visible = false;
    //
    // label20
    //
    this->label20->AutoSize = true;
    this->label20->Location = System::Drawing::Point(83,

45);

    this->label20->Name = L"label20";
    this->label20->Size = System::Drawing::Size(41, 13);
    this->label20->TabIndex = 11;
    this->label20->Text = L"label20";
    this->label20->Visible = false;
    //
    // label21
    //
    this->label21->AutoSize = true;
    this->label21->Location = System::Drawing::Point(162,

45);

    this->label21->Name = L"label21";
    this->label21->Size = System::Drawing::Size(41, 13);
    this->label21->TabIndex = 12;
    this->label21->Text = L"label21";
    this->label21->Visible = false;
    //
    // label22
    //
    this->label22->AutoSize = true;
    this->label22->Location = System::Drawing::Point(4,

67);

    this->label22->Name = L"label22";
    this->label22->Size = System::Drawing::Size(41, 13);
    this->label22->TabIndex = 13;
    this->label22->Text = L"label22";

```



```

        this->label22->Visible = false;
        //
        // label24
        //
        this->label24->AutoSize = true;
        this->label24->Location = System::Drawing::Point(162,
67);

        this->label24->Name = L"label24";
        this->label24->Size = System::Drawing::Size(41, 13);
        this->label24->TabIndex = 15;
        this->label24->Text = L"label24";
        this->label24->Visible = false;
        //
        // label23
        //
        this->label23->AutoSize = true;
        this->label23->Location = System::Drawing::Point(83,
67);

        this->label23->Name = L"label23";
        this->label23->Size = System::Drawing::Size(41, 13);
        this->label23->TabIndex = 14;
        this->label23->Text = L"label23";
        this->label23->Visible = false;
        //
        // button5
        //
        this->button5->Enabled = false;
        this->button5->Location = System::Drawing::Point(660,
10);

        this->button5->Name = L"button5";
        this->button5->Size = System::Drawing::Size(110, 30);
        this->button5->TabIndex = 12;
        this->button5->Text = L"Загрузить файл";
        this->button5->UseVisualStyleBackColor = true;
        this->button5->Click += gcnew
System::EventHandler(this, &MyForm::button5_Click);
        //
        // openFileDialog1
        //
        this->openFileDialog1->FileName = L"openFileDialog1";
        this->openFileDialog1->FileOk += gcnew
System::ComponentModel::CancelEventHandler(this,
&MyForm::openFileDialog1_FileOk);
        //
        // button6
        //
        this->button6->Enabled = false;
        this->button6->Location = System::Drawing::Point(920,
10);

        this->button6->Name = L"button6";
        this->button6->Size = System::Drawing::Size(110, 30);

```

```

        this->button6->TabIndex = 13;
        this->button6->Text = L"Загрузить файл";
        this->button6->UseVisualStyleBackColor = true;
        this->button6->Click += gcnew
System::EventHandler(this, &MyForm::button6_Click);
        //
        // button7
        //
        this->button7->Enabled = false;
        this->button7->Location = System::Drawing::Point(1205,
10);

        this->button7->Name = L"button7";
        this->button7->Size = System::Drawing::Size(96, 30);
        this->button7->TabIndex = 17;
        this->button7->Text = L"Загрузить файл";
        this->button7->UseVisualStyleBackColor = true;
        this->button7->Click += gcnew
System::EventHandler(this, &MyForm::button7_Click);
        //
        // tableLayoutPanel3
        //
        this->tableLayoutPanel3->BackColor =
System::Drawing::SystemColors::Control;
        this->tableLayoutPanel3->CellBorderStyle =
System::Windows::Forms::TableLayoutPanelCellBorderStyle::Single;
        this->tableLayoutPanel3->ColumnCount = 3;
        this->tableLayoutPanel3->ColumnStyles->Add((gcnew
System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::
Percent,
                                33.33F)));
        this->tableLayoutPanel3->ColumnStyles->Add((gcnew
System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::
Percent,
                                33.33F)));
        this->tableLayoutPanel3->ColumnStyles->Add((gcnew
System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::
Percent,
                                33.34F)));
        this->tableLayoutPanel3->Controls->Add(this->label125,
0, 0);
        this->tableLayoutPanel3->Controls->Add(this->label128,
0, 1);
        this->tableLayoutPanel3->Controls->Add(this->label129,
1, 1);
        this->tableLayoutPanel3->Controls->Add(this->label130,
2, 1);
        this->tableLayoutPanel3->Controls->Add(this->label131,
0, 2);
        this->tableLayoutPanel3->Controls->Add(this->label132,
1, 2);

```

```

        this->tableLayoutPanel3->Controls->Add(this->label13,
2, 2);
        this->tableLayoutPanel3->Controls->Add(this->label14,
0, 3);
        this->tableLayoutPanel3->Controls->Add(this->label16,
2, 3);
        this->tableLayoutPanel3->Controls->Add(this->label15,
1, 3);
        this->tableLayoutPanel3->Controls->Add(this->label127,
2, 0);
        this->tableLayoutPanel3->Controls->Add(this->label126,
1, 0);
        this->tableLayoutPanel3->Location =
System::Drawing::Point(1061, 368);
        this->tableLayoutPanel3->Name = L"tableLayoutPanel3";
        this->tableLayoutPanel3->RowCount = 4;
        this->tableLayoutPanel3->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel3->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel3->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel3->RowStyles->Add((gcnew
System::Windows::Forms::RowStyle(System::Windows::Forms::SizeType::Per
cent, 25)));
        this->tableLayoutPanel3->Size =
System::Drawing::Size(240, 91);
        this->tableLayoutPanel3->TabIndex = 16;
        //
        // label125
        //
        this->label125->AutoSize = true;
        this->label125->Location = System::Drawing::Point(4,
1);
        this->label125->Name = L"label125";
        this->label125->Size = System::Drawing::Size(41, 13);
        this->label125->TabIndex = 17;
        this->label125->Text = L"label125";
        this->label125->Visible = false;
        //
        // label128
        //
        this->label128->AutoSize = true;
        this->label128->Location = System::Drawing::Point(4,
23);
        this->label128->Name = L"label128";
        this->label128->Size = System::Drawing::Size(41, 13);
        this->label128->TabIndex = 7;

```

```

this->label28->Text = L"label28";
this->label28->Visible = false;
//
// label29
//
this->label29->AutoSize = true;
this->label29->Location = System::Drawing::Point(83,
23);

this->label29->Name = L"label29";
this->label29->Size = System::Drawing::Size(41, 13);
this->label29->TabIndex = 8;
this->label29->Text = L"label29";
this->label29->Visible = false;
//
// label30
//
this->label30->AutoSize = true;
this->label30->Location = System::Drawing::Point(162,
23);

this->label30->Name = L"label30";
this->label30->Size = System::Drawing::Size(41, 13);
this->label30->TabIndex = 9;
this->label30->Text = L"label30";
this->label30->Visible = false;
//
// label31
//
this->label31->AutoSize = true;
this->label31->Location = System::Drawing::Point(4,
45);

this->label31->Name = L"label31";
this->label31->Size = System::Drawing::Size(41, 13);
this->label31->TabIndex = 10;
this->label31->Text = L"label31";
this->label31->Visible = false;
//
// label32
//
this->label32->AutoSize = true;
this->label32->Location = System::Drawing::Point(83,
45);

this->label32->Name = L"label32";
this->label32->Size = System::Drawing::Size(41, 13);
this->label32->TabIndex = 11;
this->label32->Text = L"label32";
this->label32->Visible = false;
//
// label33
//
this->label33->AutoSize = true;

```

```

45);

    this->label133->Location = System::Drawing::Point(162,

    this->label133->Name = L"label133";
    this->label133->Size = System::Drawing::Size(41, 13);
    this->label133->TabIndex = 12;
    this->label133->Text = L"label133";
    this->label133->Visible = false;
    //
    // label134
    //
    this->label134->AutoSize = true;
    this->label134->Location = System::Drawing::Point(4,

67);

    this->label134->Name = L"label134";
    this->label134->Size = System::Drawing::Size(41, 13);
    this->label134->TabIndex = 13;
    this->label134->Text = L"label134";
    this->label134->Visible = false;
    //
    // label136
    //
    this->label136->AutoSize = true;
    this->label136->Location = System::Drawing::Point(162,

67);

    this->label136->Name = L"label136";
    this->label136->Size = System::Drawing::Size(41, 13);
    this->label136->TabIndex = 14;
    this->label136->Text = L"label136";
    this->label136->Visible = false;
    //
    // label135
    //
    this->label135->AutoSize = true;
    this->label135->Location = System::Drawing::Point(83,

67);

    this->label135->Name = L"label135";
    this->label135->Size = System::Drawing::Size(41, 13);
    this->label135->TabIndex = 15;
    this->label135->Text = L"label135";
    this->label135->Visible = false;
    //
    // label127
    //
    this->label127->AutoSize = true;
    this->label127->Location = System::Drawing::Point(162,

1);

    this->label127->Name = L"label127";
    this->label127->Size = System::Drawing::Size(41, 13);
    this->label127->TabIndex = 16;
    this->label127->Text = L"label127";
    this->label127->Visible = false;

```

```

//
// label26
//
this->label26->AutoSize = true;
this->label26->Location = System::Drawing::Point(83,
1);

this->label26->Name = L"label26";
this->label26->Size = System::Drawing::Size(41, 13);
this->label26->TabIndex = 0;
this->label26->Text = L"label26";
this->label26->Visible = false;
//
// dataGridView5
//
this->dataGridView5->AutoSizeColumnsMode =
System::Windows::Forms::DataGridViewAutoSizeColumnsMode::Fill;
this->dataGridView5->ColumnHeadersHeightSizeMode =
System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoS
ize;
this->dataGridView5->Columns->AddRange(gcnew
cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {
    this->dataGridViewTextBoxColumn10,
    this->dataGridViewTextBoxColumn11, this-
>dataGridViewTextBoxColumn12
});
this->dataGridView5->Location =
System::Drawing::Point(1061, 52);
this->dataGridView5->Name = L"dataGridView5";
this->dataGridView5->RowTemplate->Height = 24;
this->dataGridView5->Size = System::Drawing::Size(240,
300);

this->dataGridView5->TabIndex = 15;
//
// dataGridViewTextBoxColumn10
//
this->dataGridViewTextBoxColumn10->HeaderText = L"R";
this->dataGridViewTextBoxColumn10->Name =
L"dataGridViewTextBoxColumn10";
this->dataGridViewTextBoxColumn10->ReadOnly = true;
//
// dataGridViewTextBoxColumn11
//
this->dataGridViewTextBoxColumn11->HeaderText = L"G";
this->dataGridViewTextBoxColumn11->Name =
L"dataGridViewTextBoxColumn11";
this->dataGridViewTextBoxColumn11->ReadOnly = true;
//
// dataGridViewTextBoxColumn12
//
this->dataGridViewTextBoxColumn12->HeaderText = L"B";

```

```

        this->dataGridViewTextBoxColumn12->Name =
L"dataGridViewTextBoxColumn12";
        this->dataGridViewTextBoxColumn12->ReadOnly = true;
        //
        // button8
        //
        this->button8->Location = System::Drawing::Point(1061,
10);

        this->button8->Name = L"button8";
        this->button8->Size = System::Drawing::Size(138, 36);
        this->button8->TabIndex = 14;
        this->button8->Text = L"Метод с учетом степени
значимости цвета";
        this->button8->UseVisualStyleBackColor = true;
        this->button8->Click += gcnew
System::EventHandler(this, &MyForm::button8_Click);
        //
        // MyForm
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(6,
13);

        this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
        this->ClientSize = System::Drawing::Size(1317, 470);
        this->Controls->Add(this->button7);
        this->Controls->Add(this->tableLayoutPanel3);
        this->Controls->Add(this->dataGridView5);
        this->Controls->Add(this->button8);
        this->Controls->Add(this->button6);
        this->Controls->Add(this->button5);
        this->Controls->Add(this->tableLayoutPanel2);
        this->Controls->Add(this->dataGridView4);
        this->Controls->Add(this->tableLayoutPanel1);
        this->Controls->Add(this->dataGridView3);
        this->Controls->Add(this->button4);
        this->Controls->Add(this->button3);
        this->Controls->Add(this->dataGridView2);
        this->Controls->Add(this->dataGridView1);
        this->Controls->Add(this->button2);
        this->Controls->Add(this->button1);
        this->Name = L"MyForm";
        this->Text = L"Калибровка";

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
->dataGridView1))->EndInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
->dataGridView2))->EndInit();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
->dataGridView3))->EndInit();

```

```

        this->tableLayoutPanel1->ResumeLayout(false);
        this->tableLayoutPanel1->PerformLayout();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
->dataGridView4))->EndInit();
        this->tableLayoutPanel2->ResumeLayout(false);
        this->tableLayoutPanel2->PerformLayout();
        this->tableLayoutPanel3->ResumeLayout(false);
        this->tableLayoutPanel3->PerformLayout();

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
->dataGridView5))->EndInit();
        this->ResumeLayout(false);

    }
#pragma endregion

    private: System::Void openFileDialog1_FileOk(System::Object^
sender, System::ComponentModel::CancelEventArgs^ e) {}

    void WriteOnGrid(int number_of_string, double r,
double g, double b, uchar number_of_grid)
    {
        switch (number_of_grid)
        {
            case 1:
                dataGridView1->Rows->Add(); //Добавляем
                dataGridView1->Rows[number_of_string]-
                >Cells[0]->Value = r.ToString();
                dataGridView1->Rows[number_of_string]-
                >Cells[1]->Value = g.ToString();
                dataGridView1->Rows[number_of_string]-
                >Cells[2]->Value = b.ToString();
                break;
            case 2:
                dataGridView2->Rows->Add(); //Добавляем
                dataGridView2->Rows[number_of_string]-
                >Cells[0]->Value = r.ToString();
                dataGridView2->Rows[number_of_string]-
                >Cells[1]->Value = g.ToString();
                dataGridView2->Rows[number_of_string]-
                >Cells[2]->Value = b.ToString();
                break;
            case 3:
                dataGridView3->Rows->Add(); //Добавляем
                dataGridView3->Rows[number_of_string]-
                >Cells[0]->Value = r.ToString();

```



```

                                dataGridView3->Rows[number_of_string]-
>Cells[1]->Value = g.ToString();
                                dataGridView3->Rows[number_of_string]-
>Cells[2]->Value = b.ToString();
                                break;
                                case 4:
                                    dataGridView4->Rows->Add(); //Добавляем
строку в GridView
                                dataGridView4->Rows[number_of_string]-
>Cells[0]->Value = r.ToString();
                                dataGridView4->Rows[number_of_string]-
>Cells[1]->Value = g.ToString();
                                dataGridView4->Rows[number_of_string]-
>Cells[2]->Value = b.ToString();
                                break;
                                case 5:
                                    dataGridView5->Rows->Add(); //Добавляем
строку в GridView
                                dataGridView5->Rows[number_of_string]-
>Cells[0]->Value = r.ToString();
                                dataGridView5->Rows[number_of_string]-
>Cells[1]->Value = g.ToString();
                                dataGridView5->Rows[number_of_string]-
>Cells[2]->Value = b.ToString();
                                break;
                                }
                            }

```

```

private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e)
{
    std::string imageName("../Images\\NoWB_285_CCD.bmp");
    colorChecker_0 = cv::imread(imageName.c_str(),
cv::IMREAD_COLOR);

    cvNamedWindow("ColorCheckerCCD", CV_WINDOW_NORMAL);
    cvSetMouseCallback("ColorCheckerCCD", myMouseCallback,
(void*)&colorChecker_0);
    cv::resizeWindow("ColorCheckerCCD", 640, 420);
    imshow("ColorCheckerCCD", colorChecker_0);
}

```

```

private: System::Void button2_Click(System::Object^ sender,
System::EventArgs^ e)
{
    std::string imageName("../Images\\NoWB_236_CMOS.bmp");
    colorChecker_P = cv::imread(imageName.c_str(),
cv::IMREAD_COLOR);

    cvNamedWindow("ColorCheckerCMOS", CV_WINDOW_NORMAL);
}

```

```

        cvSetMouseCallback("ColorCheckerCMOS", myMouseCallback2,
(void*)&colorChecker_P);
        cv::resizeWindow("ColorCheckerCMOS", 640, 420);
        imshow("ColorCheckerCMOS", colorChecker_P);
    }

```

```

private: System::Void button3_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (m_O == m_P)
    {
        double **temp; //Промежуточная обратная матрица
        if (gaus_obr(4, Multiplication(Transp(AddColumn(P,
m_O), m_O, 4), AddColumn(P, m_O), 4, m_O, 4), temp) == 1)
        {
            M12 = Multiplication(Multiplication(temp,
Transp(AddColumn(P, m_O), m_O, 4), 4, 4, m_O), 0, 4, m_O, 3);

            label1->Visible = TRUE;
            label2->Visible = TRUE;
            label3->Visible = TRUE;
            label4->Visible = TRUE;
            label5->Visible = TRUE;
            label6->Visible = TRUE;
            label7->Visible = TRUE;
            label8->Visible = TRUE;
            label9->Visible = TRUE;
            label10->Visible = TRUE;
            label11->Visible = TRUE;
            label12->Visible = TRUE;
            label1->Text = (floor(M12[0][0] * 100000 + .5) /
100000).ToString(); //Заполняем матрицу линейного преобразования цветов
            label2->Text = (floor(M12[0][1] * 100000 + .5) /
100000).ToString();
            label3->Text = (floor(M12[0][2] * 100000 + .5) /
100000).ToString();
            label4->Text = (floor(M12[1][0] * 100000 + .5) /
100000).ToString();
            label5->Text = (floor(M12[1][1] * 100000 + .5) /
100000).ToString();
            label6->Text = (floor(M12[1][2] * 100000 + .5) /
100000).ToString();
            label7->Text = (floor(M12[2][0] * 100000 + .5) /
100000).ToString();
            label8->Text = (floor(M12[2][1] * 100000 + .5) /
100000).ToString();
            label9->Text = (floor(M12[2][2] * 100000 + .5) /
100000).ToString();
            label10->Text = (floor(M12[3][0] * 100000 + .5) /
100000).ToString();

```

```

label11->Text = (floor(M12[3][1] * 100000 + .5) /
100000).ToString();
label12->Text = (floor(M12[3][2] * 100000 + .5) /
100000).ToString();
button5->Enabled = TRUE;

double ** P_mod; //Калибруемые цвета,
восстановленные с помощью матрицы линейного преобразования
P_mod = Multiplication(AddColumn(P, m_0), M12,
m_0, 4, 3);

double ** delta; //Матрица разностей между
опорными и восстановленными цветами
delta = Difference(0, P_mod, m_0, 3);

for (size_t i = 0; i < m_0; i++)
{
    WriteOnGrid(i, 0[i][0], 0[i][1], 0[i][2],
1); //Заполняем grid_1 опорными цветами
    WriteOnGrid(i, P[i][0], P[i][1], P[i][2],
2); //Заполняем grid_2 калибруемыми цветами
    WriteOnGrid(i, delta[i][0], delta[i][1],
delta[i][2], 3); //Заполняем grid_3 дельтами между опорными и
откалиброванными цветами
}
}
else
    MessageBox::Show("Не удалось взять обратную
матрицу!", "Error", MessageBoxButtons::OK,
MessageBoxIcon::Exclamation);
}
else
    MessageBox::Show("Количество опорных и калибруемых
цветов не совпадает! " + (m_0 + 1).ToString() +
"!=" + (m_P + 1).ToString(), "Error",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
}
private: System::Void button4_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (m_0 == m_P)
    {
        double **temp; //Промежуточная обратная матрица
        if (gaus_obr(4, Multiplication(Transp(AddColumn(P,
m_0), m_0, 4), AddColumn(P, m_0), 4, m_0, 4), temp) == 1)
        {
            M12_mod = Multiplication(Multiplication(temp,
Transp(AddColumn(P, m_0), m_0, 4), 4, 4, m_0), 0, 4, m_0, 3);

            double ** P_mod; //Калибруемые цвета,
восстановленные с помощью матрицы линейного преобразования

```

```

        double ** delta; //Матрица разностей между
опорными и восстановленными цветами
        P_mod = Multiplication(AddColumn(P, m_0),
M12_mod, m_0, 4, 3);
        delta = Difference(0, P_mod, m_0, 3);

        double **M12_previous = new double*[4];
        for (size_t i = 0; i < 4; i++)
            M12_previous[i] = new double[3];

        bool flag = false;
        while (flag == false)
        {
            double *E = new double[m_0]; //Вектор ошибки
            for (size_t i = 0; i < m_0; i++)
            {
                E[i] = 0;
                for (size_t j = 0; j < 3; j++)
                    E[i] = E[i] + pow(delta[i][j],
2);

                E[i] = sqrt(E[i]);
            }

            double sum = 0;
            double *c = new double[m_0]; //Вектор весов
            for (size_t i = 0; i < m_0; i++)
            {
                c[i] = 1 / (E[i] + 0.1);
                sum = sum + pow(c[i], 2);
            }
            for (size_t i = 0; i < m_0; i++)
                c[i] = c[i] / sqrt(sum); //Нормируем
вектор весов

            double **C = new double*[m_0]; //Формируем
весовую матрицу

            for (size_t i = 0; i < m_0; i++)
            {
                C[i] = new double[m_0];
                for (size_t j = 0; j < m_0; j++)
                    C[i][j] = 0;
                C[i][i] = pow(c[i], 2);
            }

            if (gaus_obr(4,
Multiplication(Multiplication(Transp(AddColumn(P, m_0), m_0, 4), C, 4,
m_0, m_0), AddColumn(P, m_0), 4, m_0, 4), temp) == 1)
            {
                for (size_t i = 0; i < 4; i++)
                    for (size_t j = 0; j < 3; j++)

```

```

M12_mod[i][j];

M12_previous[i][j] =

M12_mod =
Multiplication(Multiplication(Multiplication(temp, Transp(AddColumn(P,
m_0), m_0, 4), 4, 4, m_0), C, 4, m_0, m_0), 0, 4, m_0, 3);

P_mod = Multiplication(AddColumn(P,
m_0), M12_mod, m_0, 4, 3);
delta = Difference(0, P_mod, m_0, 3);

flag = true;
for (size_t i = 0; i < 4; i++)
//Заканчиваем итерации
for (size_t j = 0; j < 3; j++)
if (abs(M12_mod[i][j] -
M12_previous[i][j]) < err)
flag = flag & true;
else
flag = false;
}
else
MessageBox::Show("Не удалось взять
обратную матрицу!", "Error", MessageBoxButtons::OK,
MessageBoxIcon::Exclamation);
}

for (size_t i = 0; i < m_0; i++)
{
label13->Visible = TRUE;
label14->Visible = TRUE;
label15->Visible = TRUE;
label16->Visible = TRUE;
label17->Visible = TRUE;
label18->Visible = TRUE;
label19->Visible = TRUE;
label20->Visible = TRUE;
label21->Visible = TRUE;
label22->Visible = TRUE;
label23->Visible = TRUE;
label24->Visible = TRUE;
label13->Text = (floor(M12_mod[0][0] *
100000 + .5) / 100000).ToString(); //Заполняем матрицу линейного
преобразования цветов
label14->Text = (floor(M12_mod[0][1] *
100000 + .5) / 100000).ToString();
label15->Text = (floor(M12_mod[0][2] *
100000 + .5) / 100000).ToString();
label16->Text = (floor(M12_mod[1][0] *
100000 + .5) / 100000).ToString();

```

```

        label17->Text = (floor(M12_mod[1][1] *
100000 + .5) / 100000).ToString();
        label18->Text = (floor(M12_mod[1][2] *
100000 + .5) / 100000).ToString();
        label19->Text = (floor(M12_mod[2][0] *
100000 + .5) / 100000).ToString();
        label20->Text = (floor(M12_mod[2][1] *
100000 + .5) / 100000).ToString();
        label21->Text = (floor(M12_mod[2][2] *
100000 + .5) / 100000).ToString();
        label22->Text = (floor(M12_mod[3][0] *
100000 + .5) / 100000).ToString();
        label23->Text = (floor(M12_mod[3][1] *
100000 + .5) / 100000).ToString();
        label24->Text = (floor(M12_mod[3][2] *
100000 + .5) / 100000).ToString();
        button6->Enabled = TRUE;

        WriteOnGrid(i, O[i][0], O[i][1], O[i][2],
1); //Заполняем grid_1 опорными цветами
        WriteOnGrid(i, P[i][0], P[i][1], P[i][2],
2); //Заполняем grid_2 калибруемыми цветами
        WriteOnGrid(i, delta[i][0], delta[i][1],
delta[i][2], 4); //Заполняем grid_4 дельтами между опорными и
откалиброванными цветами
    }
    }
    else
        MessageBox::Show("Не удалось взять обратную
матрицу!", "Error", MessageBoxButtons::OK,
MessageBoxIcon::Exclamation);
    }
    else
        MessageBox::Show("Количество опорных и калибруемых
цветов не совпадает! " + (m_0 + 1).ToString() +
"!=" + (m_P + 1).ToString(), "Error",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
    }
private: System::Void button5_Click(System::Object^ sender,
System::EventArgs^ e)
{
    std::string imageName;
    openFileDialog1->InitialDirectory = "\\Images";
    openFileDialog1->Filter =
"image(*.jpg,*bmp,*png)|*.jpg;*.bmp;*.png|All files (*.*)|*.*";
    openFileDialog1->Title = "Выбор файла";
    if (openFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK) //Открываем диалоговое окно
открытия файла
    {
        System::IO::StreamReader ^ sr = gcnew

```

```

        System::IO::StreamReader(openFileDialog1->
>FileName);
        imageName =
msclr::interop::marshal_as<std::string>(openFileDialog1->FileName);
        sr->Close();
    }

    cv::Mat image = cv::imread(imageName.c_str(),
cv::IMREAD_COLOR); // Загружаем изображение в класс Mat

    cvNamedWindow("До калибровки", CV_WINDOW_NORMAL);
    cv::resizeWindow("До калибровки", 640, 420);
    imshow("До калибровки", image);

    cv::Mat image_mod = image;
    uchar *input = (uchar*)(image.data);
    uchar *input2 = (uchar*)(image_mod.data);
    int cn = image.channels();

    for (int i = 0; i < image.rows; i++)
    {
        for (int j = 0; j < image.cols; j++)
        {
            input2[i*image_mod.cols*cn + j*cn + 2] =
M12[0][0] + input[i*image.cols*cn + j*cn + 2] * M12[1][0] +
            input[i*image.cols*cn + j*cn + 1] *
M12[2][0] + input[i*image.cols*cn + j*cn + 0] * M12[3][0] + 0.5;
            input2[i*image_mod.cols*cn + j*cn + 1] =
M12[0][1] + input[i*image.cols*cn + j*cn + 2] * M12[1][1] +
            input[i*image.cols*cn + j*cn + 1] *
M12[2][1] + input[i*image.cols*cn + j*cn + 0] * M12[3][1] + 0.5;
            input2[i*image_mod.cols*cn + j*cn + 0] =
M12[0][2] + input[i*image.cols*cn + j*cn + 2] * M12[1][2] +
            input[i*image.cols*cn + j*cn + 1] *
M12[2][2] + input[i*image.cols*cn + j*cn + 0] * M12[3][2] + 0.5;
        }
    }

    cvNamedWindow("После калибровки", CV_WINDOW_NORMAL);
    cv::resizeWindow("После калибровки", 640, 420);
    imshow("После калибровки", image_mod);
}

private: System::Void button6_Click(System::Object^ sender,
System::EventArgs^ e)
{
    std::string imageName;
    openFileDialog1->InitialDirectory = "\\Images";
    openFileDialog1->Filter =
"image(*.jpg,*.bmp,*.png)|*.jpg;*.bmp;*.png|All files (*.*)|*.*";
    openFileDialog1->Title = "Выбор файла";
}

```

```

        if (openFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK) //Открываем диалоговое окно
открытия файла
        {
            System::IO::StreamReader ^ sr = gcnew
                System::IO::StreamReader(openFileDialog1-
>FileName);

            imageName =
msclr::interop::marshal_as<std::string>(openFileDialog1->FileName);
            sr->Close();
        }

        cv::Mat image = cv::imread(imageName.c_str(),
cv::IMREAD_COLOR); // Загружаем изображение в класс Mat

        cvNamedWindow("До калибровки", CV_WINDOW_NORMAL);
        cv::resizeWindow("До калибровки", 640, 420);
        imshow("До калибровки", image);

        cv::Mat image_mod = image;
        uchar *input = (uchar*)(image.data);
        uchar *input2 = (uchar*)(image_mod.data);
        int cn = image.channels();

        for (int i = 0; i < image.rows; i++)
        {
            for (int j = 0; j < image.cols; j++)
            {
                input2[i*image_mod.cols*cn + j*cn + 2] =
M12_mod[0][0] + input[i*image.cols*cn + j*cn + 2] * M12_mod[1][0] +
                input[i*image.cols*cn + j*cn + 1] *
M12_mod[2][0] + input[i*image.cols*cn + j*cn + 0] * M12_mod[3][0] +
0.5;

                input2[i*image_mod.cols*cn + j*cn + 1] =
M12_mod[0][1] + input[i*image.cols*cn + j*cn + 2] * M12_mod[1][1] +
                input[i*image.cols*cn + j*cn + 1] *
M12_mod[2][1] + input[i*image.cols*cn + j*cn + 0] * M12_mod[3][1] +
0.5;

                input2[i*image_mod.cols*cn + j*cn + 0] =
M12_mod[0][2] + input[i*image.cols*cn + j*cn + 2] * M12_mod[1][2] +
                input[i*image.cols*cn + j*cn + 1] *
M12_mod[2][2] + input[i*image.cols*cn + j*cn + 0] * M12_mod[3][2] +
0.5;
            }
        }

        cvNamedWindow("После калибровки", CV_WINDOW_NORMAL);
        cv::resizeWindow("После калибровки", 640, 420);
        imshow("После калибровки", image_mod);
    }

```



```

private: System::Void button8_Click(System::Object^ sender,
System::EventArgs^ e) {
    if (m_0 == m_P)
    {
        double **temp; //Промежуточная обратная матрица
        if (gaus_obr(4, Multiplication(Transp(AddColumn(P,
m_0), m_0, 4), AddColumn(P, m_0), 4, m_0, 4), temp) == 1)
        {
            M12_w = Multiplication(Multiplication(temp,
Transp(AddColumn(P, m_0), m_0, 4), 4, 4, m_0), 0, 4, m_0, 3);

            double ** P_mod; //Калибруемые цвета,
восстановленные с помощью матрицы линейного преобразования
            double ** delta; //Матрица разностей между
опорными и восстановленными цветами
            P_mod = Multiplication(AddColumn(P, m_0), M12_w,
m_0, 4, 3);

            delta = Difference(0, P_mod, m_0, 3);

            double **M12_prev = new double*[4];
            for (size_t i = 0; i < 4; i++)
                M12_prev[i] = new double[3];
            double *sample = new double[m_0];

            sample = normal(m_dist, s_dist);

            {
                double sum = 0;
                for (size_t i = 0; i < m_0; i++)
                {
                    sum = sum + pow(sample[i], 2);
                }
                for (size_t i = 0; i < m_0; i++)
                    sample[i] = sample[i] / sqrt(sum);
            }

            //Нормируем вектор весов

            double **C = new double*[m_0]; //Формируем
весовую матрицу

            for (size_t i = 0; i < m_0; i++)
            {
                C[i] = new double[m_0];
                for (size_t j = 0; j < m_0; j++)
                    C[i][j] = 0;
                C[i][i] = pow(sample[i], 2);
            }

            if (gaus_obr(4,
Multiplication(Multiplication(Transp(AddColumn(P, m_0), m_0, 4), C, 4,
m_0, m_0), AddColumn(P, m_0), 4, m_0, 4), temp) == 1)
            {

```

```

        for (size_t i = 0; i < 4; i++)
            for (size_t j = 0; j < 3; j++)
                M12_prev[i][j] =
M12_w[i][j];

        M12_w =
Multiplication(Multiplication(Multiplication(temp, Transp(AddColumn(P,
m_0), m_0, 4), 4, 4, m_0), C, 4, m_0, m_0), 0, 4, m_0, 3);

        P_mod = Multiplication(AddColumn(P,
m_0), M12_w, m_0, 4, 3);
        delta = Difference(0, P_mod, m_0, 3);
    }
    else
        MessageBox::Show("Не удалось взять
обратную матрицу!", "Error", MessageBoxButtons::OK,
MessageBoxIcon::Exclamation);
    }

    for (size_t i = 0; i < m_0; i++)
    {
        label25->Visible = TRUE;
        label26->Visible = TRUE;
        label27->Visible = TRUE;
        label28->Visible = TRUE;
        label29->Visible = TRUE;
        label30->Visible = TRUE;
        label31->Visible = TRUE;
        label32->Visible = TRUE;
        label33->Visible = TRUE;
        label34->Visible = TRUE;
        label35->Visible = TRUE;
        label36->Visible = TRUE;
        label25->Text = (floor(M12_w[0][0] * 100000
+ .5) / 100000).ToString(); //Заполняем матрицу линейного
преобразования цветов
        label26->Text = (floor(M12_w[0][1] * 100000
+ .5) / 100000).ToString();
        label27->Text = (floor(M12_w[0][2] * 100000
+ .5) / 100000).ToString();
        label28->Text = (floor(M12_w[1][0] * 100000
+ .5) / 100000).ToString();
        label29->Text = (floor(M12_w[1][1] * 100000
+ .5) / 100000).ToString();
        label30->Text = (floor(M12_w[1][2] * 100000
+ .5) / 100000).ToString();
        label31->Text = (floor(M12_w[2][0] * 100000
+ .5) / 100000).ToString();
        label32->Text = (floor(M12_w[2][1] * 100000
+ .5) / 100000).ToString();
    }

```

```

label13->Text = (floor(M12_w[2][2] * 100000
+ .5) / 100000).ToString();
label14->Text = (floor(M12_w[3][0] * 100000
+ .5) / 100000).ToString();
label15->Text = (floor(M12_w[3][1] * 100000
+ .5) / 100000).ToString();
label16->Text = (floor(M12_w[3][2] * 100000
+ .5) / 100000).ToString();
button7->Enabled = TRUE;

WriteOnGrid(i, O[i][0], O[i][1], O[i][2],
1); //Заполняем grid_1 опорными цветами
WriteOnGrid(i, P[i][0], P[i][1], P[i][2],
2); //Заполняем grid_2 калибруемыми цветами
WriteOnGrid(i, delta[i][0], delta[i][1],
delta[i][2], 5); //Заполняем grid_5 дельтами между опорными и
откалиброванными цветами
    }
}
else
    MessageBox::Show("Не удалось взять обратную
матрицу!", "Error", MessageBoxButtons::OK,
MessageBoxIcon::Exclamation);
}
else
    MessageBox::Show("Количество опорных и калибруемых
цветов не совпадает! " + (m_O + 1).ToString() +
"!=" + (m_P + 1).ToString(), "Error",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
}
private: System::Void button7_Click(System::Object^ sender,
System::EventArgs^ e) {
    std::string imageName;
    openFileDialog1->InitialDirectory = "\\Images";
    openFileDialog1->Filter =
"image(*.jpg;*.bmp;*.png)|*.jpg;*.bmp;*.png|All files (*.*)|*.*";
    openFileDialog1->Title = "Выбор файла";
    if (openFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK) //Открываем диалоговое окно
открытия файла
    {
        System::IO::StreamReader ^ sr = gcnew
        System::IO::StreamReader(openFileDialog1-
>FileName);
        imageName =
mscorlib::interop::marshal_as<std::string>(openFileDialog1->FileName);
        sr->Close();
    }

    cv::Mat image = cv::imread(imageName.c_str(),
cv::IMREAD_COLOR); // Загружаем изображение в класс Mat

```

```

cvNamedWindow("До калибровки", CV_WINDOW_NORMAL);
cv::resizeWindow("До калибровки", 640, 420);
imshow("До калибровки", image);

cv::Mat image_mod = image;
uchar *input = (uchar*)(image.data);
uchar *input2 = (uchar*)(image_mod.data);
int cn = image.channels();

for (int i = 0; i < image.rows; i++)
{
    for (int j = 0; j < image.cols; j++)
    {
        input2[i*image_mod.cols*cn + j*cn + 2] =
M12_w[0][0] + input[i*image.cols*cn + j*cn + 2] * M12_w[1][0] +
        input[i*image.cols*cn + j*cn + 1] *
M12_w[2][0] + input[i*image.cols*cn + j*cn + 0] * M12_w[3][0] + 0.5;
        input2[i*image_mod.cols*cn + j*cn + 1] =
M12_w[0][1] + input[i*image.cols*cn + j*cn + 2] * M12_w[1][1] +
        input[i*image.cols*cn + j*cn + 1] *
M12_w[2][1] + input[i*image.cols*cn + j*cn + 0] * M12_w[3][1] + 0.5;
        input2[i*image_mod.cols*cn + j*cn + 0] =
M12_w[0][2] + input[i*image.cols*cn + j*cn + 2] * M12_w[1][2] +
        input[i*image.cols*cn + j*cn + 1] *
M12_w[2][2] + input[i*image.cols*cn + j*cn + 0] * M12_w[3][2] + 0.5;
    }
}

cvNamedWindow("После калибровки", CV_WINDOW_NORMAL);
cv::resizeWindow("После калибровки", 640, 420);
imshow("После калибровки", image_mod);
}

private: System::Void dataGridView3_CellContentClick(System::Object^
sender, System::Windows::Forms::DataGridViewCellEventArgs^ e) {
}
};

double Get_Color(uchar **image, int size) //Усредняем цвет по
окну размером [size, size]
{
    double sum = 0;
    for (size_t i = 0; i < size; i++)
        for (size_t j = 0; j < size; j++)
            sum = sum + image[i][j];
    sum = sum / (size * size);

    for (int i = 0; i < size; i++)
        delete image[i]; //освобождает память, выделенную для
массива значений

```

```

        delete[] image; //освобождает память, выделенную под массив
указателей
        return sum;
    }

    void myMouseCallback(int event, int x, int y, int flags, void*
param)
    {
        cv::Mat matImg(*(cv::Mat*)param); //Конвертируем *param в
класс Mat
        cv::Mat roi_orig; //Mat для оригинального окна

        uchar **b; //Двумерные динамические массивы, содержащие
информацию о цветовых компонентах пикселей
        uchar **g;
        uchar **r;
        uchar *input;

        int x_, y_; //Координаты x и y левого верхнего края окна

        x_ = x - roi_size / 2;
        y_ = y - roi_size / 2;
        if (x - roi_size / 2 < 0)
            x_ = 0;
        if (x + roi_size / 2 > matImg.cols)
            x_ = matImg.cols - roi_size;
        if (y - roi_size / 2 < 0)
            y_ = 0;
        if (y + roi_size / 2 > matImg.rows)
            y_ = matImg.rows - roi_size;

        cv::Rect roi = cv::Rect(x_, y_, roi_size, roi_size);
        //Выделяем область для обработки

        switch (event) {
            case CV_EVENT_MOUSEMOVE:
                break;

            case CV_EVENT_LBUTTONDOWN:
                matImg(roi).copyTo(roi_orig); //Копируем область
интересов из большого изображения в маленькое ROI
                cvNamedWindow("Образец цвета опорного изображения",
CV_WINDOW_NORMAL); //Создаем окно для отображения изображения
                cv::resizeWindow("Образец цвета опорного изображения",
roi_size, roi_size); //Меняем размеры окна
                imshow("Образец цвета опорного изображения",
roi_orig); //Отображаем окно

                input = (uchar*)(roi_orig.data);

```

```

        b = new uchar*[roi_orig.rows]; //Двумерные
динамические массивы, содержащие информацию о цветовых компонентах
пикселей
        g = new uchar*[roi_orig.rows];
        r = new uchar*[roi_orig.rows];

        for (int i = 0; i < roi_orig.rows; i++) {
//Преобразуем класс Mat в двумерный динамический массив uchar
            b[i] = new uchar[roi_orig.cols];
            g[i] = new uchar[roi_orig.cols];
            r[i] = new uchar[roi_orig.cols];
            for (int j = 0; j < roi_orig.cols; j++) {
                b[i][j] = input[roi_orig.step * i + 3 * j];
                g[i][j] = input[roi_orig.step * i + 3 * j +
1];
                r[i][j] = input[roi_orig.step * i + 3 * j +
2];
            }
            m_0++;
            O = (double**)realloc((void *)O, m_0 *
sizeof(double*)); //Добавляем строку
            O[m_0 - 1] = (double*)malloc(3 * sizeof(double));
//Выделяем память под указатели для столбцов к новой строке*/
            O[m_0 - 1][0] = Get_Color(r, roi_size); //Получаем
усредненные значения цвета и записываем их в массив
            O[m_0 - 1][1] = Get_Color(g, roi_size);
            O[m_0 - 1][2] = Get_Color(b, roi_size);

            break;

        case CV_EVENT_LBUTTONDOWN:
            break;
        }
    }

    void myMouseCallback2(int event, int x, int y, int flags, void*
param)
    {
        cv::Mat matImg(*(cv::Mat*)param); //Конвертируем *param в
класс Mat
        cv::Mat roi_orig; //Mat для оригинального окна

        uchar **b; //Двумерные динамические массивы, содержащие
информацию о цветовых компонентах пикселей
        uchar **g;
        uchar **r;
        uchar *input;

        int x_, y_; //Координаты x и y левого верхнего края окна

```

```

x_ = x - roi_size / 2;
y_ = y - roi_size / 2;
if (x - roi_size / 2 < 0)
    x_ = 0;
if (x + roi_size / 2 > matImg.cols)
    x_ = matImg.cols - roi_size;
if (y - roi_size / 2 < 0)
    y_ = 0;
if (y + roi_size / 2 > matImg.rows)
    y_ = matImg.rows - roi_size;
cv::Rect roi = cv::Rect(x_, y_, roi_size, roi_size); //Выделяем
область для обработки

switch (event) {
case CV_EVENT_MOUSEMOVE:
    break;

case CV_EVENT_LBUTTONDOWN:
    matImg(roi).copyTo(roi_orig); //Копируем область
интересов из большого изображения в маленькое ROI
    cvNamedWindow("Образец цвета калибруемого
изображения", CV_WINDOW_NORMAL); //Создаем окно для отображения
изображения
    cv::resizeWindow("Образец цвета калибруемого
изображения", roi_size, roi_size); //Меняем размеры окна
    imshow("Образец цвета калибруемого изображения",
roi_orig); //Отображаем окно

    input = (uchar*)(roi_orig.data);
    b = new uchar*[roi_orig.rows]; //Двумерные
динамические массивы, содержащие информацию о цветовых компонентах
пикселей

    g = new uchar*[roi_orig.rows];
    r = new uchar*[roi_orig.rows];

    for (int i = 0; i < roi_orig.rows; i++) {
//Преобразуем класс Mat в двумерный динамический массив uchar
        b[i] = new uchar[roi_orig.cols];
        g[i] = new uchar[roi_orig.cols];
        r[i] = new uchar[roi_orig.cols];
        for (int j = 0; j < roi_orig.cols; j++) {
            b[i][j] = input[roi_orig.step * i + 3 * j];
            g[i][j] = input[roi_orig.step * i + 3 * j +
1];
            r[i][j] = input[roi_orig.step * i + 3 * j +
2];
        }
    }
    m_P++;

```

```

        P = (double**)realloc((void *)P, m_P *
sizeof(double*)); //Добавляем строку

        P[m_P - 1] = (double*)malloc(3 * sizeof(double));
//Выделяем память под указатели для столбцов к новой строке*/
        P[m_P - 1][0] = Get_Color(r, roi_size); //Получаем
усредненные значения цвета и записываем их в массив
        P[m_P - 1][1] = Get_Color(g, roi_size);
        P[m_P - 1][2] = Get_Color(b, roi_size);
        break;

    case CV_EVENT_LBUTTONDOWN:
        break;
    }
}

double **AddColumn(double **matrix_input, int m) //Добавление
единичного столбца слева
{
    double **matrix_output = new double*[m];
    for (size_t i = 0; i < m; i++)
        matrix_output[i] = new double[4];

    for (size_t i = 0; i < m; i++)
    {
        matrix_output[i][0] = 1;
        for (size_t j = 0; j < 3; j++)
        {
            matrix_output[i][j + 1] = matrix_input[i][j];
        }
    }
    return matrix_output;
}

int gaus_obr(int cnt_str, double **mass, double **&M_obr) //Вычисление
обратной матрицы
{
    int i, j, k;
    //создание единичной матрицы
    M_obr = new double*[cnt_str];
    for (i = 0; i < cnt_str; i++)
    {
        M_obr[i] = new double[cnt_str];
        for (j = 0; j < cnt_str; j++) M_obr[i][j] = 0;
        M_obr[i][i] = 1;
    }
    //прямой ход методом Гаусса
    double a, b;
    for (i = 0; i < cnt_str; i++)
    {
        a = mass[i][i];

```



```

        for (j = i + 1; j < cnt_str; j++)
        {
            b = mass[j][i];
            for (k = 0; k < cnt_str; k++)
            {
                mass[j][k] = mass[i][k] * b - mass[j][k] *
a;
                M_obr[j][k] = M_obr[i][k] * b - M_obr[j][k]
* a;
            }
        }
    }
    //обратный ход нахождения элементов обратной матрицы
    double sum;
    for (i = 0; i < cnt_str; i++)
    {
        for (j = cnt_str - 1; j >= 0; j--)
        {
            sum = 0;
            for (k = cnt_str - 1; k > j; k--)
                sum += mass[j][k] * M_obr[k][i];
            if (mass[j][j] == 0)
            {
                for (i = 0; i < cnt_str; i++)
                    delete[] M_obr[i];
                delete[] M_obr;
                return 0;
            }
            M_obr[j][i] = (M_obr[j][i] - sum) / mass[j][j];
        }
    }
    return 1;
}

double ** Transp(double **matrix_input, int m, int n)
//Транспонирование матрицы
{
    double **matrix_output = new double*[n];
    for (size_t i = 0; i < n; i++)
        matrix_output[i] = new double[m];

    for (size_t i = 0; i < n; i++)
        for (size_t j = 0; j < m; j++)
            matrix_output[i][j] = matrix_input[j][i];

    return matrix_output;
}

double ** Multiplication(double **matrix_1, double **matrix_2, int m,
int n, int p) //Перемножение матриц
{
    double **matrix_output = new double*[m];

```

```

        for (size_t i = 0; i < m; i++)
            matrix_output[i] = new double[p];

        for (size_t k = 0; k < p; k++)
            for (size_t i = 0; i < m; i++)
            {
                matrix_output[i][k] = 0;
                for (size_t j = 0; j < n; j++)
                    matrix_output[i][k] = matrix_output[i][k] +
matrix_1[i][j] * matrix_2[j][k];
            }
        return matrix_output;
    }

double ** Difference(double **matrix_1, double **matrix_2, int m, int
n) //Вычитание матриц
{
    double **matrix_output = new double*[m];
    for (size_t i = 0; i < m; i++)
        matrix_output[i] = new double[n];

    for (size_t i = 0; i < m; i++)
        for (size_t j = 0; j < n; j++)
            matrix_output[i][j] = matrix_1[i][j] -
matrix_2[i][j];

    return matrix_output;
}

double * normal(const double m, const double s) //функция нормального
распределения весовых коэффициентов
{
    std::random_device rd;
    std::mt19937 gen(rd());
    std::normal_distribution<> distr(m, s);
    std::vector<double> v;
    for (int i = 0; i < m_0; i++)
    {
        double temp = distr(gen); //генерируем значения из
норм распределения
        v.push_back(temp); //вносим значения в вектор
    }

    sort(v.rbegin(), v.rend()); //сортируем вектор в порядке
убывания

    double *samples = new double[m_0];
    for (int i = 0; i < m_0; i++)
        samples[i] = v[i]; //вносим значения из вектора в
массив sample
    return samples;
}

```

} }