Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Системное программное обеспечение вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Утилита для поиска дубликатов изображений

Студент: гр. 150504 Бура В. Э.

Руководитель: Кавальчук А. Н.

Минск 2014

[ВВЕДЕНИЕ 4](#_Toc390103921)

[1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ 5](#_Toc390103922)

[1.1 Алгоритм обнаружения углов Моравека 6](#_Toc390103923)

[1.2 Детектор Харриса 6](#_Toc390103924)

[1.3 Масштабируемый детектор Харриса-Лапласа 7](#_Toc390103925)

[1.4 Алгоритм поиска углов Ши-Томаси 9](#_Toc390103926)

[1.5 SIFT 10](#_Toc390103927)

[1.6 SURF 11](#_Toc390103928)

[2 ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ АЛГОРИТМОВ 14](#_Toc390103929)

[2.1 Поиск ключевых точек 14](#_Toc390103930)

[2.2 Построение дескриптора 15](#_Toc390103931)

[2.3 Сопоставление ключевых точек 18](#_Toc390103932)

[3 РЕАЛИЗАЦИЯ ПРОГРАММНОЙ ЧАСТИ КУРСОВОГО ПРОЕКТА 19](#_Toc390103933)

[3.1 Функция harrisDetector() 19](#_Toc390103934)

[3.2 Функция laplaceDetector() 19](#_Toc390103935)

[3.3 Функция computeMainOrient() 20](#_Toc390103936)

[3.4 Функция matchKeypoints() 21](#_Toc390103937)

[ОПТИМИЗАЦИЯ 22](#_Toc390103938)

[4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 24](#_Toc390103939)

[5 ТЕСТИРОВАНИЕ 25](#_Toc390103940)

[ЗАКЛЮЧЕНИЕ 28](#_Toc390103941)

[ЛИТЕРАТУРА 29](#_Toc390103942)

[ПРИЛОЖЕНИЕ А 30](#_Toc390103943)

[ИСХОДНЫЙ КОД ФУНКЦИИ harrisDetector() 30](#_Toc390103944)

[ПРИЛОЖЕНИЕ B 31](#_Toc390103945)

[ИСХОДНЫЙ КОД ФУНКЦИИ laplaceDetector() 31](#_Toc390103946)

[ПРИЛОЖЕНИЕ С 32](#_Toc390103947)

[ИСХОДНЫЙ КОД ФУНКЦИИ computeMainOrient() 32](#_Toc390103948)

# ВВЕДЕНИЕ

В настоящее время объем обрабатываемой человеком информации неуклонно растет. Это приводит к необходимости автоматизации процессов, которые ранее выполнялись вручную. Одними из основных задач в автоматической обработке информации являются:

* поиск объектов определенного класса (например, поиск лиц/пешеходов на изображениях);
* сравнение объектов внутри классов (распознавание лиц в видеопотоке или на изображении, поиск по отпечаткам пальцев, поиск дубликатов изображений, сравнение голосов).

Данный курсовой проект посвящен второй из упомянутых задач, а именно — поиску дубликатов изображений.

По некоторым подсчетам количество изображений в сети Интернет в настоящее время достигает 1012 и с каждым годом увеличивается в геометрической прогрессии. Совершенно очевидно, что производить поиск на такой выборке вручную не представляется возможным.

Также, можно прогнозировать, что по мере оцифровки архивов и библиотек, значимость поиска дубликатов изображений будет возрастать. Хотя бы потому, что большую часть новой информации человек получает через зрительную систему[2].

# 1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

Поиск дубликатов изображений базируется на поиске соответствий — одной из основных задач машинного зрения. Она базируется на процессе выделения и сопоставления особенностей изображений.

Важной задачей на этапе выделения особенностей является описание свойств объекта — дескрипторов, с помощью которых в дальнейшем будут проводится сопоставление и идентификация.

Дескриптор, или вектор признаков — набор численных параметров особенности , например цвет, текстура и т.д. Простейшим примером дескриптора может служить окрестность особенности, записанная в виде вектора. Все дескрипторы принимают значения в каком-либо пространстве признаков. Следовательно, если задать меру близости в этом пространстве, то дескрипторы можно будет сравнивать[1].

Дескрипторы можно разделить на две группы:

* глобальные дескрипторы
* локальные дескрипторы

Под глобальными дескрипторами понимаются такие вектора признаков, которые были получены в результате анализа изображения в целом. Обычно при применении глобальных дескрипторов каждая точка изображения вносит вклад (больший или меньший) в значение формируемых дескрипторов.

Локальные дескрипторы — это собственно и есть дескрипторы точечных особенностей. Они описывают какие-то выделяющиеся части изображения, по которым оно может быть выделено из множества других.

Учитывая вышесказанное, можно сделать вывод, что сферы применения глобальных и локальных дескрипторов различны. Глобальные дескрипторы применяются для поиска по общему подобию, т.е. семантически похожих с точки зрения человека. Локальные дескрипторы применяются при поиске кропнутых изображений, а также при поиске изображений по фрагменту. Стоит отметить, на сегодняшний день методы, использующие локальные дескрипторы, показывают лучшие результаты, поэтому при разработке алгоритмов для курсового проекта было решено использовать локальные дескрипторы.

Однако перед тем, как рассмотреть существующие методы построения локальных дескрипторов, остановимся на алгоритмах нахождения ключевых точек изображения. Следует отметить, что некоторые дескрипторы одновременно являются и детекторами особых точек.

## Алгоритм обнаружения углов Моравека

Это один из первых алгоритмов обнаружения углов и определяет угол, чтобы тот был в точке с низкой автомодельностью. Алгоритм проверяет каждый пиксель в изображении, чтобы определить является ли тот углом, рассматривая участки в области пикселя. Сходство определяется путем принятия суммы квадратов разностей между двумя участками. Меньшее число указывает на большее сходство[5].

Если пиксель в области равномерной интенсивностью, то близлежащие участки будут выглядеть примерно одинаково. Если пиксель находится на краю, тогда соседние участки в направлении, перпендикулярном к краю будет выглядеть совершенно разными, но соседние участки в направлении, параллельном краю изменяются незначительно. Если пиксель на особенности с изменением во всех направлениях, то ни один из близлежащих участков не будет выглядеть примерно также.

Сила угла определяется как наименьшая сумма квадратов разностей между участком и его соседями (по горизонтали, вертикали и двум диагоналям). Если это число локально максимально, то особенность присутствует.

Как отметил Моравек, одна из главных проблем этого метода связана с тем, что он не изотропен: если угол не направлен в сторону соседей, то он не будет обнаружен, как точечная особенность[5].

## Детектор Харриса

Харрис и Стивенс улучшили детектор углов Моравека, рассматривая дифференциальную оценку угла по отношению к направлению непосредственно, вместо использования сдвинутых пятен. Эту оценку угла часто называют автокорреляционной, поскольку этот термин используется в том документе, в котором этот детектор описан. Однако с математической точки зрения используется метод суммы квадратов разностей[5].

Без потери общности будем считать, что используются полутоновые 2-мерные изображения. Пусть это изображение будет задано I. Рассмотрим вопрос о выделении области изображения (U, V) и перехода его по (х, у). Взвешенную сумму квадратов разностей между этими двумя областями, обозначим S, определяющуюся по формуле:

(1.1)

может быть аппроксимирована рядом Тейлора. Пусть будут частными производными от , такими что

(1.2)

Это приводит к приближению

(1.3)

которое можно записать в матричном виде

(1.4)

где А — структура тензора

(1.5)

Эта матрица — матрица Харриса. Угол (или, в общем, точечная особенность) характеризуется большим изменением S во всех направлениях вектора . На основе анализа собственных значений A, эта характеристика может быть выражена следующим образом: должно быть два "больших" собственных значения для точечных особенностей. На основании величины собственных значений, можно сделать следующие выводы на основе этих аргументов:

* если   и  то этот пиксель (х, у) не имеет особенности, представляющей интерес;
* если  и имеет некоторое большое положительное значение, то обнаружен край;
* если и большие положительные значения, то угол найден.

Харрис и Стивенс отметили, что точное определение собственных значений вычислительно дорого, так как требует вычисления квадратного корня, а вместо этого предложить следующие функции с M, где κ является настраиваемым параметром чувствительности:

(1.6)

Таким образом, алгоритм не имеет на самом деле вычисления собственного разложения матрицы, а вместо этого достаточно вычислить определитель и след от A найти углы, или, вернее, точки интереса в целом.

Значение k должно быть определено эмпирически, так и в литературе были представлены как возможные значения в диапазоне 0,04 - 0,15.

## Масштабируемый детектор Харриса-Лапласа

Вычисление матрицы второго момента A в детекторе Харриса, требует вычисления производных изображения в области изображения, а также суммы нелинейных комбинаций этих производных по соседству. Так как вычисление производных обычно включает этап расширения сглаживания пространства, оперативное определение оператора Харриса требует два параметра масштабирования: в местном масштабе для сглаживания до вычисления производных изображения и степени интеграции накопления нелинейных операций по производным операторов в единый дескриптор изображения[5].

Если *I* обозначает интенсивность оригинального изображения, пусть *L* обозначает масштаб пространства представление  *I* полученное путем свертки с гауссовым ядром

(1.7)

с локальным параметром масштабирования t:

(1.8)

и пусть  и  обозначают частные производные от *L*. Кроме того, введем гауссовскую весовую функцию с интеграционным параметром масштабирования *s*. Тогда масштабируемая матрица второго момента может быть определена как

(1.9)

Теперь мы можем вычислить собственные значения μ таким же образом, как и для A, и определить масштабируемую меру углов Харриса

(1.10)

Что касается выбора местного параметра масштаба t и масштаба параметров интеграции S, эти параметры масштаба, как правило, связанных между собой относительный показатель степень интеграции таким, что , где γ обычно выбирается в интервале . Таким образом, мы можем вычислить масштабируемую меру углов Харриса в любом масштабе t в расширении пространства для получения многомасштабных детекторов углов, который реагирует на структуру углов разного размера в области изображения.

На практике этот масштабируемый детектор углов часто дополняется шагом выбора масштабирования, где масштаб-нормированный оператор Лапласа

(1.11)

рассчитывается в каждом масштабе в масштабах пространства и масштаба адаптированы угловых точек с автоматическим выбором масштаба ("Харрис-Лапласа") вычисляются по точкам, которые являются одновременно:

* пространственных максимумов многомасштабных мер углов

(1.12)

* локальных максимумов и минимумов в масштабах шкалы-нормированные оператор Лапласа

(1.13)

## Алгоритм поиска углов Ши-Томаси

Детектор углов во многом основан на угловом детекторе Харриса. Авторы показывают, что для областей изображения проходят аффинные преобразования, *min(λ1,λ2)* является более точным измерением силы угла, чем *Mc*.

Ранние подходы в обнаружении углов обнаруживали места, где кривизна линий и градиент одновременно высоки. Дифференциальный способ выявления таких точек для вычисления кривизны масштабированной кривой (произведение кривизны и градиента в степени 3)

(1.14)

и выявлять положительные максимумы и отрицательные минимумы этого дифференциального выражения в некотором масштабе *t* в масштабах пространства *L* представление исходного изображения. Основной проблемой этого подхода, является то, что он очень чувствителен к шуму и к выбору масштабного уровня. Лучший метод для расчета γ-нормированной масштабированной кривизны

(1.15)

с *γ=7/8*, а также обнаружения подписали расширения пространства максимумов этого выражения, которые являются центрами и масштабах, которые максимумов положительных и отрицательных минимумов по отношению *k* пространства и масштаба

(1.16)

в сочетании с дополнительным шагом локализации справиться с увеличением локализации ошибки на грубые весы. Таким образом, большие значения масштаба будут связаны с закругленными углами большой пространственной протяженности, в то время как меньшие значения масштаба будут связаны с острыми углами с небольшой пространственной протяженности. Такой подход является первым детектором углов с автоматическим выбором масштаба (до "Харрис-Лапласа" выше), и был использован для отслеживания углов под большие изменения масштаба в области изображения[5].

Теперь рассмотрим несколько видов собственно дескрипторов особых точек

## SIFT

Метод SIFT (Scale Invariant Feature Transform) обнаруживает и описывает локальные особенности изображения. Получаемые с помощью него признаки инвариантны относительно масштаба и поворота, устойчивы к ряду аффинных преобразований, шуму[4].

Алгоритм SIFT является достаточно эффективным, но он обладает высокой сложностью обработки. Данный алгоритм можно разделить на две части: определение «точек интереса» (key points, points of interest) и построение дескрипторов окрестностей данных точек.

Алгоритм, предложенный в рамках SIFT, заключается в использования пирамиды Гаусса, которая строится для изображения. Далее изображения приводятся к одному размеру, и вычисляется их разность (DoG, difference-of-Gaussian images). Причем в качестве кандидатов точек интереса выбираются только те пиксели, которые сильно отличаются от остальных, это делается, например, путем сравнения каждого пикселя изображения с несколькими соседними данного масштаба, с несколькими соответствующими соседями в большем и меньшем масштабе. Пиксель выбирается как точка интереса только в том случае, если его яркость является экстремумом[4].

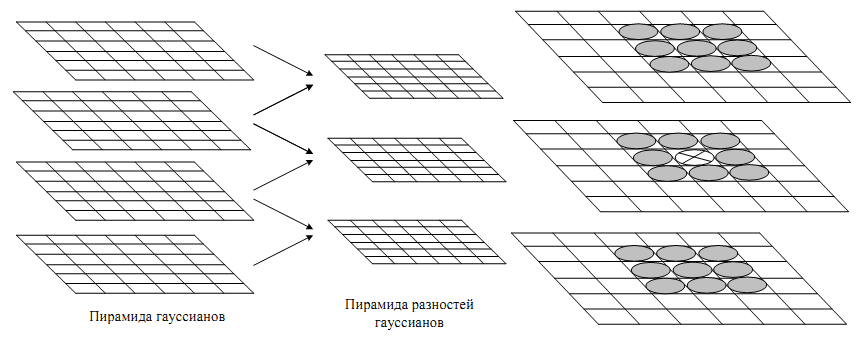


Рисунок 1.1 — Нахождение ключевых точек SIFT

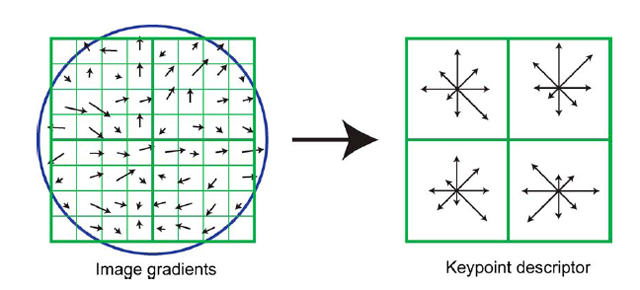


Рисунок 1.2 — Построение гистограммы градиентов SIFT

Для описания особенности используются значения и направления локальных градиентов пикселов из окрестности ключевой точки. Значения градиентов суммируются с весами, равными значениям Гауссиана с центром в ключевой точке, как показано на рисунке 1.2. Далее окрестность разбивается на 4 квадрата, и для каждого строится гистограмма. Таким образом, на каждую ключевую точку расходуется 4\*4\*8 = 128 байт[4].

SIFT дескрипторы не лишены недостатков. Не все полученные точки и их дескрипторы будут отвечать предъявляемым требованиям. И это будет сказываться на дальнейшем решении задачи сопоставления изображений.

Метод не работает в следующих случаях:

* Условия освещения различны (например, день / ночь).
* Объект имеет отражающую поверхность (как правило, автомобили, зеркала).
* Объект имеет выраженную 3-D структуру.
* Объект имеет себе подобные структуры: тогда попадаются «истинные» несоответствия.
* Угол обзора слишком отличается.

Также добавим, что данный метод запатентован, что затрудняет его использованию в коммерческих проектах.

## SURF

Альтернативным подходом является SURF (Speeded Up RobustFeatures), который в несколько раз быстрее SIFT. В данном подходе для ускорения поиска точек интереса используются интегральные изображения. С помощью интегральных изображений за константное время вычисляются так называемые прямоугольные фильтры, которые состоят из нескольких прямоугольных областей.

SURF решает две задачи – поиск особых точек изображения и создание их дескрипторов, инвариантных к масштабу и вращению[3].

Определение особых точек на изображении выполняется на основании матрицы Гессе (FAST-Hessian detector). Использование Гессиана обеспечивает инвариантность относительно преобразования типа «поворот», но не инвариантность относительно изменения масштаба. Поэтому SURF применяет фильтры разного масштаба для вычисления Гессиана.

Детерминант матрицы Гессе достигает экстремума в точках максимального изменения градиента яркости. Далее для каждой найденной особой точки вычисляется ориентация – преобладающее направление перепада яркости. Понятие ориентации близко к понятию направления градиента, но для определения ориентации особой точки применяется фильтр Хаара.

После нахождения ключевых точек, SURF формирует их дескрипторы:

* Вокруг точки строится квадратная окрестность размером 20s, где s – масштаб, на котором получено максимальное значение детерминанта матрицы Гесса.
* Полученная квадратная область разбивается на блоки, в результате область будет разбита на 4x4 региона.
* Для каждого блока вычисляются более простые признаки. Как следствие, получается вектор, содержащий 4 компоненты: 2 – это суммарный градиент по квадранту, 2 – сумма модулей точечных градиентов.
* Дескриптор формируется в результате склеивания взвешенных описаний градиента для 16 квадрантов вокруг особой точки. Элементы дескриптора взвешиваются на коэффициенты Гауссова ядра.
* Дополнительно к дескриптору заносится след матрицы Гессе. Эти компоненты необходимы, чтобы различать темные и светлые пятна. Для светлых точек на темном фоне след отрицателен, для темных точек на светлом фоне – положителен.

Недостатком метода является то, что SURF используется для поиска объектов на изображении, он сам работает не с объектами. SURF никак не выделяет объект из фона. Он рассматривает изображение как единое целое и ищет особенности этого изображения. Метод слабо чувствителен к шуму[3].

Достоинствами метода SURF являются хорошие результаты при вращении и изменении освещенности изображения. К тому же он работает быстрее SIFT.

При анализе алгоритмов нахождения точечных особенностей и различных методов построения дескрипторов (в записку включено описание только небольшой их части), для реализации в курсовом проекте был выбран масштабируемый детектор Харриса-Лапласа, т.к. это один из лучших инвариантных к масштабу детекторов, и к тому же он может быть крайне легко и эффективно реализован. Дескриптор ключевых точек было решено разработать самостоятельно.

# 2 ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ АЛГОРИТМОВ

Как уже было сказано в разделе 1, поиск дубликатов можно разделить на две основные части: поиск ключевых точек изображения и построения дескрипторов для них. Однако также важной частью решаемой задачи является сопоставление ключевых точек разных изображений между собой для их дальнейшего сравнения. В данном разделе будут рассмотрены все эти этапы процесса поиска дубликатов.

## Поиск ключевых точек

Для поиска ключевых точек был выбран масштабируемый детектор Харриса-Лапласа. Используемый в данном проекте детектор несколько отличается от описанного в разделе 1, поэтому приведем алгоритм его работы:

* Вычислить значения адаптированной к масштабированию функции Харриса для масштабов :

(2.1)

(2.2)

* Количество слоев *n* и значение шага масштаба следует выбирать в зависимости от того насколько большими могут быть отличия в размерах изображений. В данном проекте были выбраны следующие значения: , , *n = 6.*
* Для каждого уровня масштаба найти локальные максимумы вычисленной функции Харриса, это и есть особые точки для данного масштаба изображения. Обычно, получается достаточно много точек и часть из них можно отбросить. Например, можно отбросить все точки, для которых значение функции Харриса не превосходит некоторого значения *Hthr,* т.к. максимумы с небольшим значением функции Харриса менее устойчивы. Я использовал *Hthr* = *0.05\*max(H).*
* Для каждой найденной таким образом особенности установить достигается ли в ней максимум функции

(2.3)

по *n,* т.е. и

Если локальный максимум не достигается, то точка отбрасывается.

* Все оставшиеся точки являются особенностями изображения, с каждой точкой ассоциирован масштаб , на котором она была обнаружена.

Пример работы детектора можно увидеть на рисунке 2.1.Красные окружности отражают радиус особенности, причем , где *r* — радиус особенности, — масштаб особенности[1].

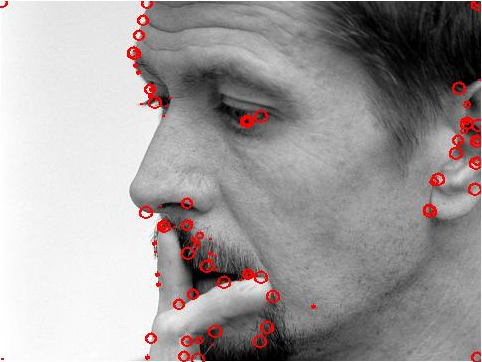


Рисунок 2.1 — Пример работы модифицированного

детектора Харриса-Лапласа

## Построение дескриптора

Построение собственного дескриптора ключевых точек является одной из главных целей курсового проекта. Разрабатываемый дескриптор должен отвечать следующим требованиям:

* Инвариантность к масштабу;
* Инвариантность к повороту;
* Инвариантность к аффинным преобразованиям;
* Вычисление за минимальное количество операций;
* Быстрота сравнения дескрипторов должна быть максимальна;
* Компактность при хранении в памяти вычислительной машины.

После анализа данных требований было решено взять за основу бинарный дескриптор BRIEF, прежде всего за счет возможности быстрого сравнения дескрипторов между собой. Теперь рассмотрим подробнее вышеуказанные требования.

1. **Инвариантность к масштабу**

При использовании масштабируемого детектора Харриса-Лапласа добиться инвариантности к изменению масштаба очень просто. Для этого достаточно провести нормировку в соответствии с локальным масштабом особенности, например если с особенностью ассоциирован масштаб *2*, то окрестность особенности следует масштабировать с коэффициентом *0,5.* Это свойство используется при сопоставлении дескрипторов: координаты ключевых точек изображения нормируются в соответствии с уровнем масштабирования, на котором они были обнаружены[1].

1. **Инвариантность к повороту**

Самый простой метод добиться инвариантности к повороту при сопоставлении особенностей — использовать дескрипторы, компоненты которых инварианты к повороту. Далее приводится пример дескриптора из [1], каждая компонента которого инвариантна к повороту. Все производные в выражении – нормированные производные (рассчитаны для значений масштаба, на котором была обнаружена ключевая точка):

(2.4)

Серьезным недостатком этого метода является то, что в дескрипторе нельзя использовать компоненты, которые не инвариантны к повороту, а число операторов, которые инвариантны к повороту, и при этом применимы на практике, ограничено.

Еще одни способ добиться инвариантности к повороту — предварительно нормировать окрестность точки особым образом, чтобы скомпенсировать поворот, и лишь потом вычислять дескрипторы для особенности. Для того чтобы нормировать окрестность по повороту, требуется оценить т.н. ориентацию особенности. В примере на рисунке 2.2 красным показаны направления локальных градиентов, синим — ориентация ключевой точки. Существует много методов оценки локальной ориентации особенности, все так или иначе используют направление векторов градиентов в окрестности особенности[1].

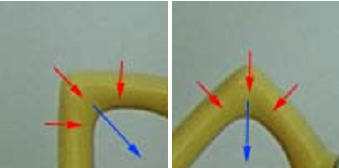


Рисунок 2.2 — Ориентация особенности

Кратко приведем алгоритм определения ориентации особенности, использованный в данном курсовой проекте:

* Разделим множество углов от 0 до 360 градусов на бины с шагом 10 градусов;
* Для каждой найденной ключевой точки:
* вычислим фазу градиента в каждой точке ее окрестности;
* построим гистограмму направлений градиентов окрестности;
* За ориентацию особенности примем угол , где *i —* индекс элемента гистограммы с наибольшим значением;

После того, как вычислена ориентация особенностей, производится их нормировка, т.е. поворот окрестности по часовой стрелке вокруг ключевой точки на угол, равный ориентации особенности.

Инвариантность к аффинным преобразованиям

К сожалению, данного требования добиться не удалось.

1. **Вычисление за минимальное количество операций**

Как и в дескрипторах BRIEF, разрабатываемый дескриптор представляет собой 64 битную строку. Значение каждого бита вычисляется следующим образом:

* берется окрестность точки размерами , где — масштаб особенности
* из окрестности выбираются пары точек , для которых строится набор бинарных тестов

(2.5)

* на основании этих тестов строится бинарная строка

(2.6)

которая и является дескриптором данной ключевой точки.

Как видим, при вычислении дескриптора используются операции сравнения и побитового сдвига, это позволяет реализовать данную функцию очень эффективно, по сравнению, например, с методом SIFT.

1. **Быстрота сравнения дескрипторов должна быть максимальна**

Как уже видно из предыдущего пункта, в качестве дескрипторов выступают бинарные строки одинаковой длины. Поэтому самым эффективным и быстрым способом сравнить их между собой является использования расстояния Хэмминга, которое также может быть эффективно реализовано на современных вычислительных машинах.

1. **Компактность при хранении в памяти вычислительной машины**

Бинарный дескриптор также решает и эту задачу. При размере дескриптора в 64 бита и среднем количестве ключевых точек на 1 изображении около 100 (эмпирические данные, полученные в ходе написания проекта) количество памяти, необходимой для хранения описания изображения, равно 1Кбайт. Учитывая, что размер среднего изображения 512x512 приблизительно равен 200Кбайт, то издержки на хранение служебной информации являются ничтожными.

## Сопоставление ключевых точек

Для сопоставления ключевых точек была выбрана Евклидова метрика. Это было сделано по следующей причине: главными характеристиками ключевой точки, характеризующими ее геометрическое положение на изображении, являются декартовы координаты ее центра. А для вычисления геометрического расстояния в многомерном пространстве Евклидова метрика показывает одни из лучших результатов.

# 3 РЕАЛИЗАЦИЯ ПРОГРАММНОЙ ЧАСТИ КУРСОВОГО ПРОЕКТА

Программная часть курсового проекта представляет собой набор скриптов и функций для среды моделирования MATLAB. Сделаем краткий обзор некоторых из них

## Функция harrisDetector()

В функции реализована первая часть детектора Харриса-Лапласа — масштабируемый детектор Харриса.

Входные параметры:

* массив *sigmaArray —* массив масштабов, на которых будет вестись поиск ключевых точек
* матрицы *xDer, yDer —* пирамиды Гауссианов для входного изображения по переменным *x* и *y* соответвенно

Выходные параметры:

* матрица *harrisPoints* — трехмерная матрица обнаруженных ключевых точек. В первые два столбца заносятся координаты ключевой точки, в последний — масштаб, на котором она была обнаружена

Краткое описание

Основную часть функции занимает цикл по всем масштабам, переданным в *sigmaArray*. В начале вычисляются элементы матрицы автокорреляции с помощью вызова функции *computeAutoCorrelationMatrix*. В нее передаются соответсвующие данному масштабу Гауссианы *xDer* и *yDer*. Затем по формуле (2.1) вычисляются значения функции Харриса для каждой точки изображения. Эти значения передаются в функциюfindLocalMax, которая осуществляет поиск локальных максимумов в окрестности, равной 3\*currSigma, где currSigma — текущий масштаб, на котором проводится поиск. Полученные точки фильтруются по порогу. Точки, значения которых превосходят порог, заносятся в выходной массив harrisPoints.

## Функция laplaceDetector()

Функция реализует второй этап детектора Харриса-Лапласа — фильтрация точек с помощью оператора Лапласа

Входные параметры:

* массив *sigmaArray —* массив масштабов, на которых будет вестись поиск ключевых точек
* матрица *harrisPoints —* точки, подозрительные на ключевые, найденные с помощью масштабируемого детектора Харриса
* матрица image *—* исходное изображение, переведенное в градации серого

Выходные параметры:

* матрица keypoints — трехмерная матрица обнаруженных ключевых точек. В первые два столбца заносятся координаты ключевой точки, в последний — масштаб, на котором она была обнаружена.

Краткое описание

Основную часть функции занимает цикл по всем элементам матрицы *harrisPoints.* Для каждой точки из harrisPoints вычисляется значение оператора Лапласа. Это значение затем проверяется на локальный максимум по формуле (2.3). Если локальный максимум достигается, точка переносится в выходную матрицу keypoints.

## Функция computeMainOrient()

Функция выполняет оценку ориентации ключевых точек

Входные параметры:

* массив sigmaArray *—* массив масштабов, на которых будет вестись поиск ключевых точек
* матрица keypoints *—* ключевые точки изображения, найденные детектором Харриса-Лапласа
* матрицы xDer*,* yDer *—* пирамиды Гауссианов для входного изображения по переменным *x* и *y* соответвенно
* матрица image *—* исходное изображение, переведенное в градации серого

Выходные параметры:

* матрица orientedKeypoints — четырехмерная матрица ключевых точек. Первые три столбца совпадают с матрицей keypoints, в четвертый заносится ориентация ключевой точки

Краткое описание

Основную функциональность в функции заключена в двух вложенных циклах. Перед этими циклами производится подготовительные операции: расширение Гауссианов на значения, равное максимальному радиусу окрестности у ключевых точек и смещение координат ключевых точек на это же значение.

Основной цикл — цикл по всем масштабам ключевых точек. Для каждого масштаба все ключевые точки, обнаруженные на нем, заносятся в матрицу kpInCurrScale и вычисляется размер окрестности для данного масштаба. Во вложенном цикле происходит вызов вспомогательной функции findKPOrient, которой передаются текущая ключевая точка, размер окрестности точки и Гауссианы для данного масштаба. Функция вычисляет и возвращает ориентацию переданной ключевой точки в градусах, которое сохраняется в выходную матрицу orientedKeypoints.

После формирования выходной матрицы производится обратный сдвиг координат ключевых точек.

## Функция matchKeypoints()

Функция сопоставляет ключевые точки двух изображений между собой, используя Евклидову метрику

Входные параметры:

* матрицы desc1 desc2 *—* матрицы ключевых точек первого и второго изображений
* x1, y1, x2, y2 *—* размеры первого и второго изображений соответственно

Выходные параметры:

* матрица descPairs — двухмерная матрица, первый столбец которой — индекс ключевой точки первого изображения, второй столбец — индекс ключевой точки второго изображения, соответствующей первой.

Краткое описание

Функция выполняет сопоставление, используя right-left matching. Для этого выполняется сопоставление для первого изображения со вторым, затем для второго с первым. В выходную матрицу заносятся только те пары, которые являются результатами обоих сопоставлений.

Но перед тем, как выполнять сопоставление требуется провести нормировку координат особенностей в соответствии с масштабом и ориентацией особенности. Эту операцию выполняет функция normCoords. После нормировки выполняется собственно сопоставление. Для этого разработана функция matchLeft, которая выполняет левое сопоставление. Затем полученные результаты объединяются и формируется выходная матрица descPairs

Реализация остальных функций и скриптов является тривиальной и не представляет интереса, поэтому их описание опустим, упомянем только назначение:

* Функция extractDescriptors — выполняет извлечение дескриптора из окрестности ключевой точки
* Функция findLocalMax — находит локальные максимумы функции в заданной окрестности
* Функция findKeypoints — «фукнция-обертка» для функций harrisDetector, laplaceDetector, computeMainOrient
* Функция computeGaussianPyramid — строит пирамиду Гауссианов изображения
* Функция computeMatchingRate — подсчитывает величину сходства между изображениями
* Функция computeAutoCorrelationMatrix — вычисляет элементы матрицы автокорреляции
* Функция buildDescriptor — выполняет собственно построение дескриптора
* Функция computeHammingDistance — вычисляет расстояние Хэмминга между двумя векторами
* Функция makeTestPatterns — формирует координаты точек для бинарных тестов при построении дескриптора

# ОПТИМИЗАЦИЯ

Среда MATLAB предоставляет удобный профайлер, который помог провести небольшую оптимизацию написанного кода.

Во-первых, как показало тестирование, выделение функции findKPOrient стало ошибкой. Время, затрачиваемое на подсчет ориентации особенностей, возросло в почти в 10 раз. Это показано на рисунке 3.1. Данное время затрачивалось на копирование Гауссианов, передаваемых в функцию. Поэтому было решено, вернуть операции, выполнявшиеся в функции findKPOrient, в функцию compMainOrient, пожертвовав красотой кода, но выиграв в производительности.

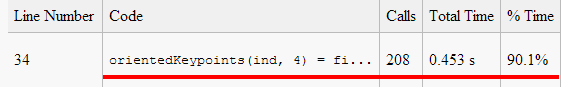




Рисунок 3.1 — Пример оптимизации исходного кода

Во-вторых, было замечено, что применение встроенной функции MATLAB для свертки conv2() при вычислении матрицы автокорреляции работает намного медленнее функции imfilter() с теме же параметрами. Работа программы также ускорилась.

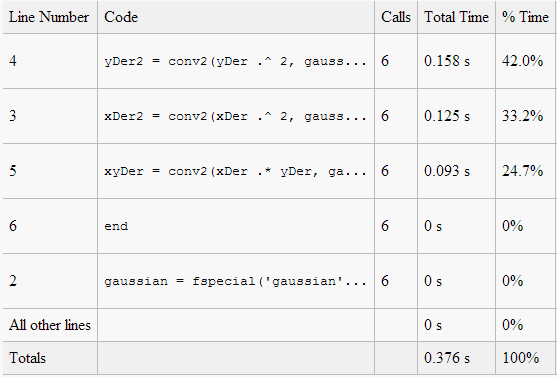


Рисунок 3.2а — Пример оптимизации кода

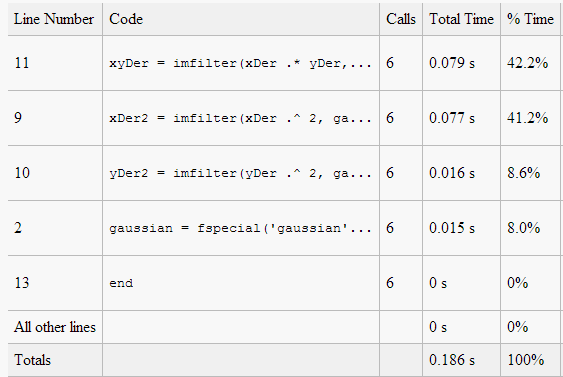


Рисунок 3.2б — Пример оптимизации кода

# 4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска утилиты необходимо наличие среды моделирования MATLAB с установленным ImageProcessing Toolbox.

Для того, чтобы проверить на сходство два изображения, нужно из командного окна вызвать функцию test, передав ей в качестве параметров названия файлов изображений. Результаты работы будут выведены в отдельном окне. В нем будут выведены входные изображения, а также результат работы программы (степень совпадения).

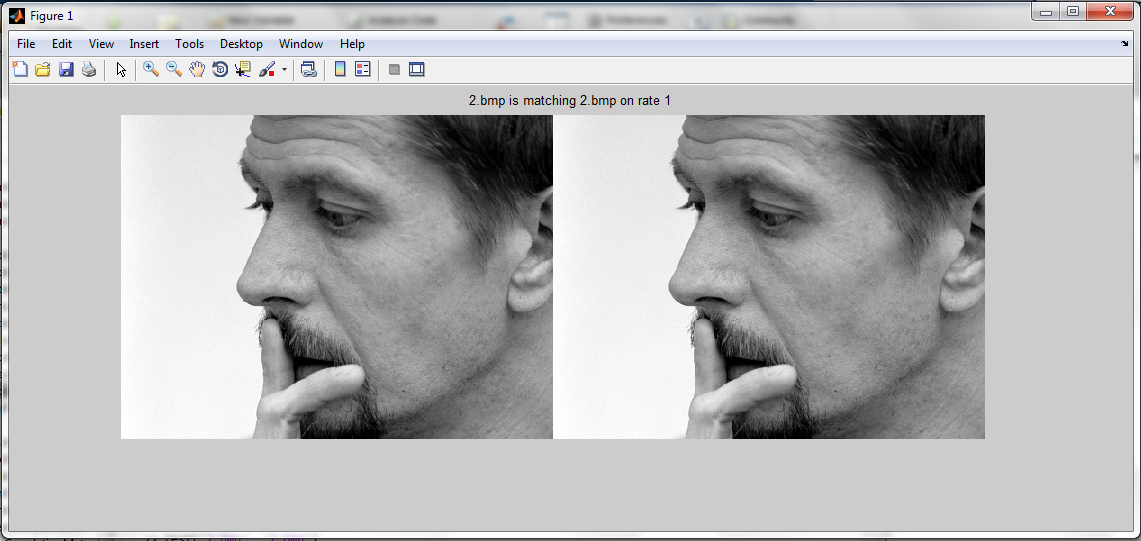


Рисунок 4.1 — Пример работы программы

# 

# 5 ТЕСТИРОВАНИЕ

Тестирование проводилось на ноутбуке с процессором Intel Core i3-2310m (2.10GHz) и 4Gb RAM. Операционная система Windows7 64bit.

Для тестирования использовались изображения лица, подвергнутого различным преобразованиям (поворот, масштабирование, инвертирование цветов). Результаты тестирования приведены ниже.

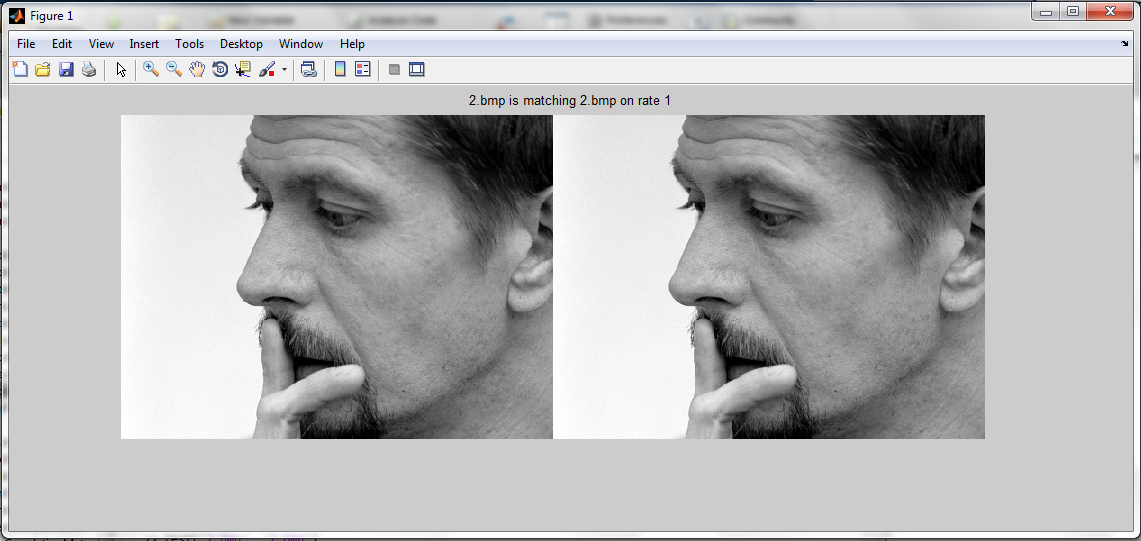


Рисунок 5.1 — Пример работы программы

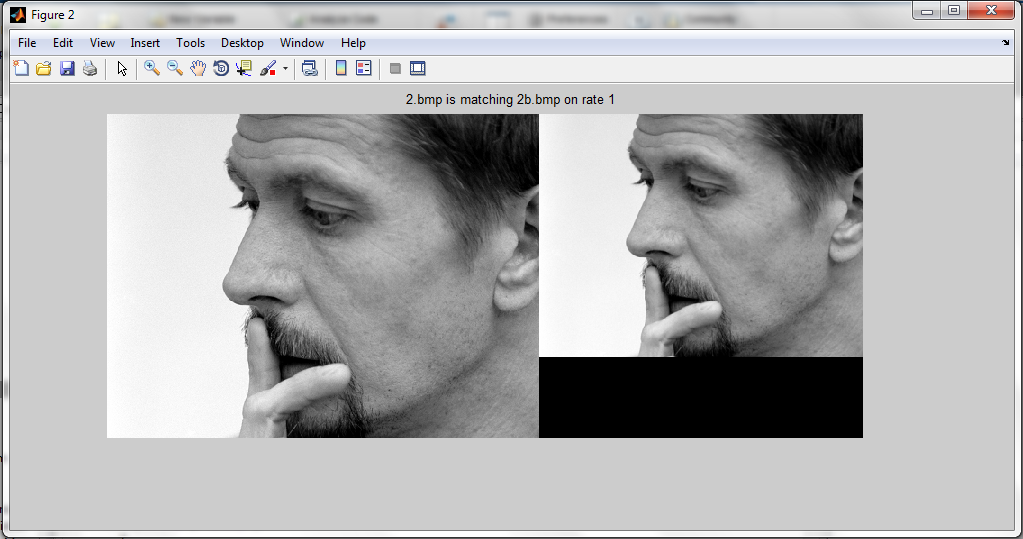


Рисунок 5.2 — Пример работы программы при масштабировании изображения

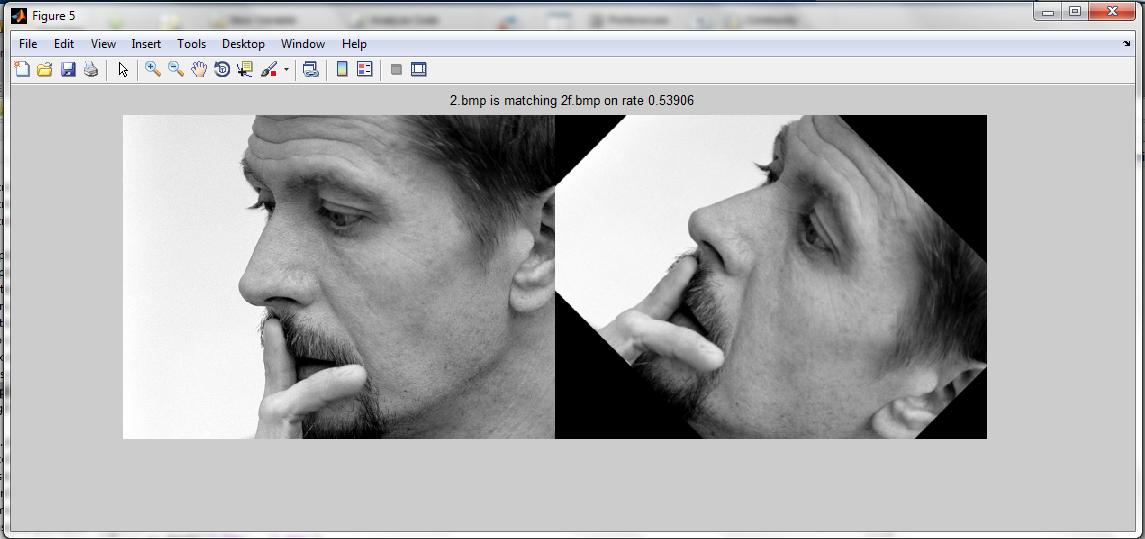


Рисунок 5.3а — Пример работы программы при повороте изображения

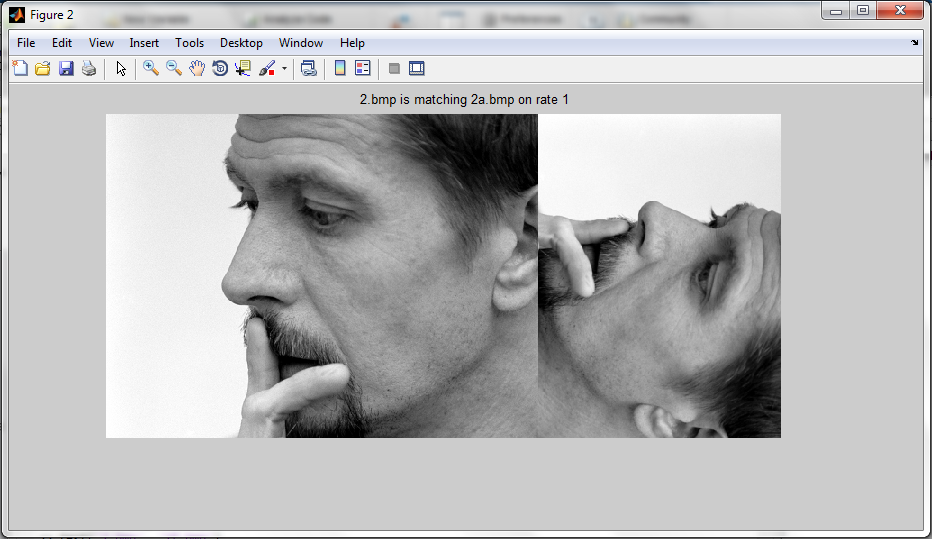


Рисунок 5.3б — Пример работы программы при повороте изображения

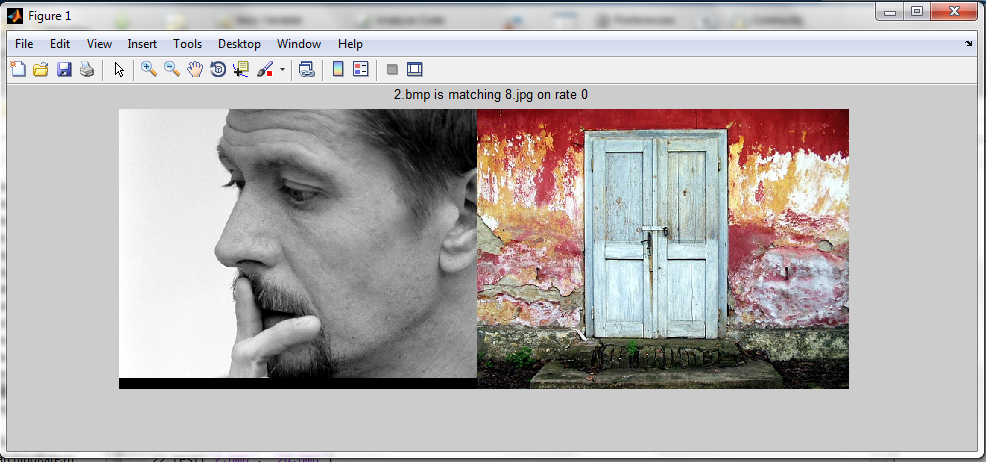


Рисунок 5.4 — Пример работы программы для разных изображений

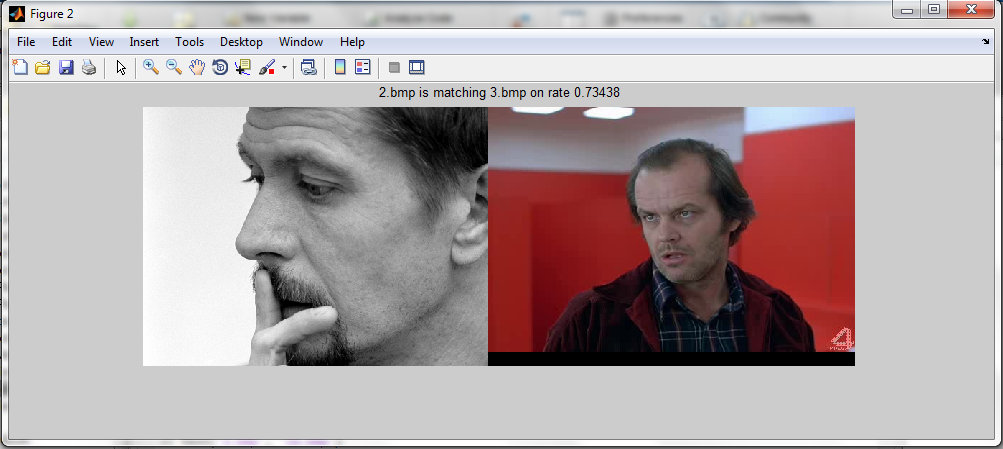


Рисунок 5.5 — Пример ложного срабатывания

Как видно, программа показывает не лучшие результаты. Также оставляет желать лучшего и время работы: для изображения размерами 480х360 пикселей полное время работы составляет ~2.2 секунд. Однако следует заметить, что значительную часть времени работы занимает вызов mex-функции morphmex, используемой при нахождении локальных максимумов. При обнаружении этой проблемы, была предпринята попытка исключить вызовы этой функции, однако, к сожалению, во всех методах обнаружения локальных максимумов в среде MATLAB в большей или меньшей степени используется morphmex.

Подробнее о том, что не получилось и как это исправить, коснемся в Заключении.

# ЗАКЛЮЧЕНИЕ

В данном курсовом проекте была создана утилита для поиска дубликатов изображений. В качестве детектора особых точек использовался масштабируемый детектор Харриса-Лапласа, дескриптор ключевых точек был разработан самостоятельно под влиянием дескриптора BRIEF.

Программа создавалась как базис для дальнейшей разработки более сложных систем, поэтому ее функционал ограничивается нахождением степени похожести между двумя поданными на вход изображениями.

Как показало тестирование, программа неплохо справляется с поворотами и масштабированием изображений. Изображения с аффинными преобразованиями обрабатываются плохо, т.к.

* используется детектор Харриса-Лапласа, который плохо находит точки при аффинных искажениях
* используется дескриптор, не учитывающий аффинные преобразования

Поэтому в данный момент при поиске вероятность правильно определить дубликат остается невысокой.

Вторым недостатком разработанной программы является время ее работы: как уже упоминалось в разделе 5, для изображения размерами 480х360 пикселей полное время работы составляет ~2.2 секунд, что не позволяет использовать ее для работы в реальном времени, например в веб-приложении. Однако следует, отметить, что при поиске изображения по базе данных, время значительно уменьшится, как будет исключен этапы нахождения ключевых точек и построения дескрипторов для них, которые как раз и занимают большую часть времени.

Исходя из вышесказанного, можно обозначить основные направления дальнейшей разработки и оптимизации:

* повышение качества поиска (прежде всего за счет использования алгоритмов, инвариантных к аффинным преобразованиям)
* уменьшение времени работы программы (в первую очередь, за изменения алгоритма поиска локальных максимумов)

Также в ходе написания курсового проекта было начато знакомство со средой MATLAB.

# ЛИТЕРАТУРА

1. Инвариантные алгоритмы сопоставления точечных особенностей на изображениях: Гаганов В. — Журнал Графика и Мультимедиа, выпуск 1(17)/2009
2. Алгоритмы поиска изображений в базах видеоданных: Десятников И.Е., Утробин В.А. — Нижегородский государственный технический университет им. Р.Е. Алексеева, 2011
3. SURF: Speeded Up Robust Features: H. Bay, T. Tuytelaars, Luc Van Gool — ETH Zurich, 2008
4. Distinctive Image Features from Scale-Invariant Keypoints: D. Lowe — CS Department Univercity of BC, 2004
5. Методы поиска угловых особенностей на изображениях [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://www.moluch.ru/archive/28/3253/>

# ПРИЛОЖЕНИЕ А

# ИСХОДНЫЙ КОД ФУНКЦИИ harrisDetector()

function [harrisPoints] = harrisDetector(sigmaArray, xDer, yDer)

nStages = size(sigmaArray, 2);

harrisPoints = zeros(0, 3);

for i = 1 : nStages

currSigma = sigmaArray(i);

[xDer2, yDer2, xyDer] = computeAutoCorrelationMatrix(xDer(:, :, i),...

yDer(:, :, i),...

currSigma);

points = (xDer2 .\* yDer2 - xyDer .^ 2) ./ (xDer2 + yDer2 + eps);

[~, ~, maxVal] = findLocalMax(points, 3 \* currSigma);

threshold = 0.05 \* max(maxVal(:));

[x, y] = find(maxVal >= threshold);

foundPointsN = size(x, 1);

harrisPoints(end + 1 : end + foundPointsN, :) = [x, y, repmat(i, [foundPointsN, 1])];

end

end

# ПРИЛОЖЕНИЕ B

# ИСХОДНЫЙ КОД ФУНКЦИИ laplaceDetector()

function keypoints = laplaceDetector(img, harrisPoints, sigmaArray)

nStages = size(sigmaArray, 2);

laplaceOper = zeros(size(img, 1), size(img, 2), nStages);

for i = 1 : nStages

currSigma = sigmaArray(i);

laplaceOper(:, :, i) = currSigma ^ 2 \* imfilter(img, ...

fspecial('log', floor(6 \* currSigma + 1), ...

currSigma), 'replicate');

end

harrisPointsN = size(harrisPoints, 1);

keypoints = zeros(harrisPointsN, 3);

kpIndex = 1;

for i = 1 : harrisPointsN

x = harrisPoints(i, 1);

y = harrisPoints(i, 2);

stage = harrisPoints(i, 3);

laplaceVal = laplaceOper(x, y, stage);

switch stage

case 1

if laplaceVal > laplaceOper(x, y, stage + 1)

keypoints(kpIndex, :) = harrisPoints(i, :);

kpIndex = kpIndex + 1;

end

case nStages

if laplaceVal > laplaceOper(x, y, stage - 1)

keypoints(kpIndex, :) = harrisPoints(i, :);

kpIndex = kpIndex + 1;

end

otherwise

if (laplaceVal > laplaceOper(x, y, stage - 1)) && ...

(laplaceVal > laplaceOper(x, y, stage + 1))

keypoints(kpIndex, :) = harrisPoints(i, :);

kpIndex = kpIndex + 1;

end

end

end

keypoints(kpIndex : end, :) = [];

end

# ПРИЛОЖЕНИЕ С

# ИСХОДНЫЙ КОД ФУНКЦИИ computeMainOrient()

function orientedKeypoints = computeMainOrient(keypoints, xDer, yDer)

scales = unique(keypoints(:, 3));

nScales = length(scales);

padVal = max(scales);

padVal = ceil(3 \* padVal);

binStep = 30;

histBins = 0 : binStep : 360;

xDer = padarray(xDer, [padVal padVal], 'symmetric');

yDer = padarray(yDer, [padVal padVal], 'symmetric');

keypoints(:, [1 2]) = keypoints(:, [1 2]) + padVal;

orientedKeypoints = zeros(0, 4);

ind = 1;

for i = 1 : nScales

kpInCurrScale = find(keypoints(:, 3) == scales(i));

nKeypoints = size(kpInCurrScale, 1);

offset = 3 \* scales(i);

for j = 1 : nKeypoints

currKP = keypoints(kpInCurrScale(j), [1 2 3]);

orientedKeypoints(ind, [1 2 3]) = currKP([1 2 3]);

kpNeighborhoodX = xDer(currKP(1,1) - offset : currKP(1,1) + offset, currKP(1,2));

kpNeighborhoodY = yDer(currKP(1,1), currKP(1,2) + offset : currKP(1,2) + offset);

gradAngles = rad2deg(atan2(kpNeighborhoodY, kpNeighborhoodX) + pi);

histogram = histc(gradAngles, histBins);

[~, m] = max(histogram);

kpOrient = histBins(m);

orientedKeypoints(ind, 4) = kpOrient;

ind = ind + 1;

end

end

orientedKeypoints(:, [1 2]) = orientedKeypoints(:, [1 2]) - padVal;

end