Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Рецензент
_____
«___»_____ 2017 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой
_____К.А. Домбровский
«___»_____2017 г.

Разработка программного комплекса обработки дактилоскопических узоров

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-09.04.01.2017.033 ПЗ ВКР
ЮУРГУ-09.04.01.2017.462 ПЗ ВКР

Руководитель работы,
профессор каф. «Электронные
Вычислительные машины»
_____В.Ю. Гудков
«___»_____2017 г.

Авторы работы
студенты группы КЭ-271
_____А.В. Агафонов
_____Д.С. Рожина
«___»_____2017 г.

Нормоконтролёр, ст. преп. каф.
«Электронные вычислительные
машины»
_____В.В. Лурье
«___»_____2017 г.

Челябинск 2017

Аннотация

Агафонов А.В., Рожина Д. С. Методы обработки цифровых дактилоскопических изображений: исследование и анализ полей направлений, потоков и адаптивная фильтрация. – Челябинск: ФГАОУ ВО «ЮУрГУ» (НИУ), КЭ; 2017, 80 с., 56 ил. Библиография литературы – 25 наименований, 13 листов чертежей ф. А1.

В данной работе рассматриваются методы улучшения качества изображений отпечатков пальца с целью повышения эффективности работы алгоритмов биометрической аутентификации. Фильтрация, скелетизация и выделение особых точек являются основными этапами в процессе идентификации по дактилоскопическому узору.

Цель работы – реализовать фильтрацию изображения отпечатка пальца, используя фильтр Габора, а также, скелетизацию и выделение особых точек с целью их дальнейшего применения в биометрических устройствах.

Задачи работы:

• рассмотреть существующие методы построения полей направлений, потоков, фильтрации изображений, скелетизации и поиска особых точек;

• разработать алгоритмы фильтрации изображений, полученных со сканера, скелетизации и выделения особых точек;

• проанализировать методы получения полей направлений и потоков, фильтрации, выделения остова и особых точек изображения;

При реализации проекта в качестве языка программирования для написания кода программной части системы был использован язык C#.

Оглавление

ВВЕДЕНИЕ 6
1 АНАЛИЗ РЕШАЕМОЙ ЗАДАЧИ 9
1.1 АКТУАЛЬНОСТЬ ЗАДАЧИ 9
1.2 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ЛИТЕРАТУРЫ 10
1.3 ФОРМУЛИРОВКА ЗАДАНИЯ 16
2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ 17
2.1 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ 17
2.2 ТРЕБОВАНИЯ К ПО 18
3 ПРЕДВАРИТЕЛЬНЫЙ ЭТАП 19
4. ОСНОВАНАЯ ЧАСТЬ 22
4.1 ПОСТРОЕНИЕ ПОЛЯ НАПРАВЛЕНИЙ 22
4.1.1 СГЛАЖИВАНИЕ 22
4.1.2 ГРАДИЕНТ В ТОЧКЕ 25
4.1.3 ГРАДИЕНТ В ОБЛАСТИ 27
4.1.4 ПОЛЕ НАПРАВЛЕНИЙ 30
4.2 ПОСТРОЕНИЕ ПОЛЯ ПЛОТНОСТИ 32
4.3 ПОСТРОЕНИЕ ИНФОРМАТИВНОЙ ОБЛАСТИ 39
4.3.1 ПОКАЗАТЕЛИ КАЧЕСТВА 43
4.3.1.1 ПОСТРОЕНИЕ КОГЕРЕНТНОСТИ 43
4.3.1.2 ВОЛОСЫ ВЕРОНИКИ 46
4.3.1.3 КОМПЛЕКСНЫЙ ПОКАЗАТЕЛЬ 48
4.3.1.4 ЭРОЗИЯ И ДИЛАТАЦИЯ 49
4.3.1.5 ВЫВОДЫ ПО ИНФОРМАТИВНОЙ ОБЛАСТИ 50
4.4 ФИЛЬТРАЦИЯ 50
4.4.1 СГЛАЖИВАНИЕ 54
4.4.2 ДИФФЕРЕНЦИРОВАНИЕ 55
4.4.3 ВЫВОДЫ ПО ФИЛЬТРАЦИИ 57
4.5 СКЕЛЕТИЗАЦИЯ И ВЫДЕЛЕНИЕ ОСОБЫХ ТОЧЕК 58
4.5.1 СКЕЛЕТИЗАЦИЯ 58
4.5.2 ВЫДЕЛЕНИЕ ОСОБЫХ ТОЧЕК 64

4.5.3 ВЫВОД ПО РАЗДЕЛУ 4.5 70
5 ПРОВЕРКА НА СООТВЕТСТВИЕ ТРЕБОВАНИЯМ ТЗ 71
6 ЗАКЛЮЧИТЕЛЬНЫЙ РАЗДЕЛ 74
СПИСОК ЛИТЕРАТУРЫ 80
ПРИЛОЖЕНИЯ 83
ПРИЛОЖЕНИЕ А 83
ПРИЛОЖЕНИЕ Б 84
ПРИЛОЖЕНИЕ В 89

ВВЕДЕНИЕ

Исследования в области отпечатков пальцев ведутся уже более 100 лет. Было разработано множество различных технологий в этой области и ряд из них успешно применены на практике.

Обработка изображений означает выполнение над ними различных операций с заданной целью, которая произрастает из двух основных областей её применения, которыми являются повышение качества изображений для улучшения его визуального восприятия человеком и обработка изображений для их хранения, передачи и распознавания.

Единой цельной и исчерпывающей теории по обработке и распознаванию изображений в настоящее время не существует и, под обработкой понимают множество подходов, методов и способов преобразований изображений, которые показали свою эффективность и практическую ценность для решения конкретных задач.

Цифровая обработка изображений относится к цифровой обработке сигналов, где сигналом служит изображение и представляет собой самостоятельную область знаний, которая быстро развивается и охватывает большой спектр методов, которые имеют очень широкое применение. [1, 2с]

Распознавание отпечатков пальцев является исключительно адаптивным способом идентификации и подходит для разностороннего применения и, в том числе, для объектов, где традиционно используются ключи, карты доступа и пароли. Биометрические системы уже используются в оборудовании контроля прохода, в автоматах выдачи инструментов, в складских помещениях, при оказании сетевых услуг и на многих других объектах. Даже новые смартфоны оборудованы сканером отпечатков пальцев. [2] Технология идентификации по отпечаткам пальцев используется повсеместно. По сравнению с другими методами идентификации, когда используется ключ, карта доступа, цифровой код или пароль, биометрический метод идентификации по отпечатку пальца

обеспечивает высокую степень защиты, так как идентификация с помощью отпечатка весьма надежный способ и, считается, что отпечатки пальцев у всех людей уникальны. [3]

Кроме того, данный метод идентификации широко применяется в криминалистике. Данные полученные по результатам обработки отпечатков могут значительно снизить круг подозреваемых в преступлении. По информации с отпечатков можно в высокой долей уверенности сделать заключение о поле и возрасте субъекта. Одним из таких критериев является плотность линий отпечатков пальцев. Это значимое свойство, которое отличается у мужчин и женщин, а также меняется с возрастом. [4]

Кроме того, методы обработки изображений играют значительную роль в научных исследованиях, промышленности, медицине, космических исследованиях и информационных системах. Примерами применения этих методов могут служить повышение четкости изображений, создаваемых электронным микроскопом, коррекция искажений изображений, принимаемых из космоса, автоматический анализ характера местности, автоматическое составление карт по аэрофотоснимкам.

Существует множество различных методов по обработке дактилоскопических изображений. Так как биометрические системы, связанные с идентификацией человека по отпечатку пальца все больше внедряются в повседневную жизнь человека, то исследования в этой области, в том числе вопрос о качестве распознавания, по-прежнему остаются актуальными.

В криминалистике основной метод установления идентичности отпечатков пальцев основан на соотнесении особых точек – разветвлений, завитков, окончаний линий, дельт. Для выделения указанных признаков требуется четкое изображение изображения. Поэтому применяются алгоритмы фильтрации, чтобы избавится от шумов. Следовательно, чем лучше качество изображения, тем точнее получается скелет, а соответственно и результат идентификации. [5]

Целью данной дипломной работы является разработка метода улучшения изображения дактилоскопического узора отпечатка пальца на основе фильтра

Габора, а также анализ влияния различных коэффициентов и параметров в применяемых методах на полученный результат.

В конечном итоге, по исследуемому изображению должен быть построен скелет и на нем отмечены особые точки. Данные о расположении таких точек применяются для сравнения отпечатков пальцев между собой. [6]

В практическом плане объектами исследований являются цифровые изображения отпечатков пальцев.

Для достижения поставленной цели были выполнены следующие этапы:

- анализ существующих методов построения полей направлений и полей потоков изображений, определение их преимуществ и недостатков;
- анализ существующих методов фильтрации изображений, определение их преимуществ и недостатков;
- анализ методов выделения остова бинарного изображения и выделения особых точек, выявление их преимуществ и недостатков;
- на основе полученных данных, разработать и реализовать алгоритм фильтрации изображений, выделения остова и особых точек;
- анализ полученных результатов, а также влияния значений различных параметров и коэффициентов, использующихся в основных и сопутствующих методах, на конечный результат.

1 АНАЛИЗ РЕШАЕМОЙ ЗАДАЧИ

1.1 Актуальность задачи

В настоящее время биометрические сенсоры, использующие физические и поведенческие характеристики человека, технологии идентификации личности и системы на их основе получили широкое распространение в различных областях: от допуска в помещения до электронной коммерции и государственных систем различного назначения. Наиболее распространенным методом биометрической идентификации является идентификация по отпечаткам пальцев.

Отпечатки пальцев играют все возрастающую роль в проблеме идентификации личности. Этому способствует возросшая производительность аппаратных средств, развитие стандартизации и интерес со стороны государства к системам идентификации.

Реальные отпечатки пальцев загрязнены, деформированы, смазаны, палец по поверхности может проскальзывать, иметь шрамы или ожоги. Это приводит к искажению модели отпечатка пальца и снижению точности и надежности идентификации папиллярного узора

При исследовании реальных объектов зачастую приходится принимать во внимание разнообразные неопределенные факторы, действующие на отпечаток пальцевого узора. Эти факторы могут быть связаны, например, со свойствами следообразующего материала, с деформацией пальца в момент следообразования, с дефектами кожи в виде ожогов, шрамов, складок, грязи и шелушения.

Таким образом, для того, чтобы преступить непосредственно к задаче идентификации, требуется предварительная обработка изображения. При помощи которой улучшается качество изображения, чтобы оно стало пригодным к дальнейшей идентификации.

Данная работа акцентирована на предварительную обработку дактилоскопического изображения.

1.2 Обзор предметной области и литературы

Многие исследователи, такие как, Бертран, Стокис и другие, нашли свидетельства того, что людей интересовали узоры на их руках ещё и в доисторическую эпоху. Например, в 1832 году, при раскопках кургана на острове Гавр-Инис (Франция), нашли каменные плиты, на которых были чертежи с изображением папиллярных узоров пальцев рук. Вначале учёные видели в них символы друидов, или думали, что они свидетельствуют о культе змей, поскольку представляли собой змееобразно извивающиеся кривые линии и т. д.

В 1885 году Александр Бертран в статье в журнале «Magasin pittoresque» отметил сходство этих изображений с узорами на пальцах. В 1920 году французский исследователь Стокис убедительно доказал, что бретонские изображения на скалах – это рисунки папиллярных узоров. Это была чрезвычайно разнообразная и довольно-таки полная коллекция изображений папиллярных линий человеческих ногтевых фаланг и ладоней. [5]

Развиваться наука папиллярных узорах, стала во второй половине XIX века. В течение довольно краткого периода времени разные люди в разных концах света, а порою совершенно независимо друг от друга, создали и развили прикладную науку дактилоскопии до уровня её практического применения и использования.

Широкому распространению дактилоскопии препятствовало отсутствие простой и надёжной классификации папиллярных узоров. Первый шаг на пути её создания сделал выходец из Далмации аргентинский полицейский чиновник Хуан Вуцетич (1858 – 1925). В сентябре 1891 г. он разработал десятипальцевую систему классификации дактилоскопических отпечатков, которую затем непрерывно улучшал до 1904 г.

Первые сведения о применении дактилоскопии в Аргентине относятся к 1892 г., когда по кровавым отпечаткам пальцев одна женщина была изобличена в убийстве своих двоих детей.

В 1896г. помощник комиссара английской полиции сэр Эдвард Р. Генри нашел способ приведения в порядок огромной картотеки, в которой хранились отпечатки пальцев. Его классификация папиллярных узоров оказалась настолько удачной, что используется во многих странах в неизменном виде и в наши дни, либо легла в основу других систем. Было выделено пять отдельных типов узоров: дуговой, шатровый, радиально-петлевой, ульнарно-петлевой и завитковый. Результаты исследований Генри были опубликованы в его книге «Классификация и использование отображений пальцев», изданной в Индии. [5]

Позже Хуан Вуцетич обнаружил, что с ростом объема картотеки появляются трудности поиска. Тогда Вуцетич стал искать характерные детали, чтобы подразделить отпечатки внутри самих групп. С его легкой руки в употребление было введено название нового метода идентификации – дактилоскопия, что в переводе с греческого означает «рассматривать пальцы».

На сегодняшний день, существует разнообразное множество методов обработки дактилоскопических узоров, которые могут применяться в различных сферах деятельности.

У дактилоскопического узора есть ряд свойств, виляющих на процесс идентификации.

Среди них плотность линий – значимое свойство, которое отличается у мужчин и женщин, а также меняется с возрастом, также данный параметр применим в криминалистике для определения пола подозреваемого. Во многих странах, таких как, например, Таиланд, Индия, Аргентина, проводились исследования, суть которых заключается в определении различий мужских и женских отпечатков по характеристике частоты линий. В рассматриваемых статьях: «Determination of sex difference from fingerprint ridge density in northeastern Thai teenagers» [7], «Sex determination from fingerprint ridge density» [8], «Fingerprint ridge density in the Argentinean population and its application to sex inference: A comparative study» [9], статье «Age and gender determination from finger prints using RVA and DCT coefficients»[10], приводятся методы и результаты подсчета и, делается ряд выводов, в том числе, что у женщин во всех

исследуемых группах наблюдается более тонкие и частые линии, чем у мужчин, таким образом, плотность линий отпечатка пальца может стать одним из признаков полового диморфизма.

Есть ряд современных авторов, книги и публикации которых являются эталонными в областях обработки изображений и в частности распознавания отпечатков пальцев.

На результаты их деятельности будут опираться многие решения, примененные в данной работе. Среди таких авторов можно отметить следующих.

Дэвид Мальтони – профессор Болонского университета, он провел множество исследований в области распознавания отпечатка пальца. В частности, его книга «Handbook of fingerprint recognition» является признанным трудов в области распознавания отпечатков пальцев. Идеи для построения плотности линий дактилоскопического узора частично взяты из данной работы. [1]

Авторы Гонсалес Р. и Вудс Р. выпустили фундаментальный труд «Цифровая обработка изображений» идеи алгоритма скелетизации изображений и некоторых вспомогательных методов позаимствованы из этой книги. [11]

Также отдельные этапы в методах фильтрации и построения поля направлений опирались на результаты, опубликованные в книге M. Bazen – «Fingerprint recognition» [12]

Перед началом обработки изображения отпечатка пальца, требуется изучить природу отпечатка: его строение, классификации, методы обработки.

Физические характеристики, такие как, лицо, отпечатки пальцев, голос, используют, чтобы определить личность человека в биометрических системах.

Сопоставление отпечатков пальцев благодаря простоте использования и отсутствия постороннего вмешательства и надежности на данный момент является самой удачной технологией биометрической идентификации.

Отпечаток пальца человека состоит из борозд и полосок, образующих сложный узор, уникальный для каждого человека, а потому, обеспечивающий оптимальный метод верификации. [3]

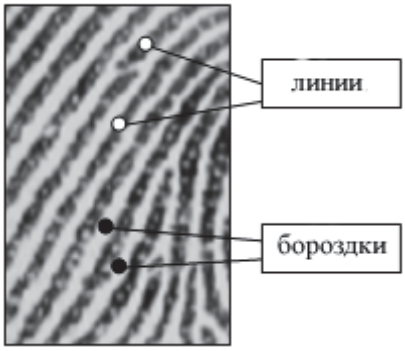


Рисунок 1.1 – Полосы и впадины на отпечатке пальца

На поверхности пальцев человека присутствуют бугорки, складывающиеся в ряды, образуя складки. Эти складки называются папиллярными линиями и отделяются друг от друга неглубокими бороздками – рисунок 1.1. Папиллярные линии на поверхности пальцев рук образуют различные узоры, называемые папиллярными узорами. Такие бороздки, сформировавшиеся на поверхности пальцев, остаются неизменными в течение всей жизни человека [13]. Из папиллярного рисунка возможно определить два типа признаков – глобальные и локальные. Глобальные признаки – это те, которые можно увидеть невооруженным глазом – рисунок 1.2.

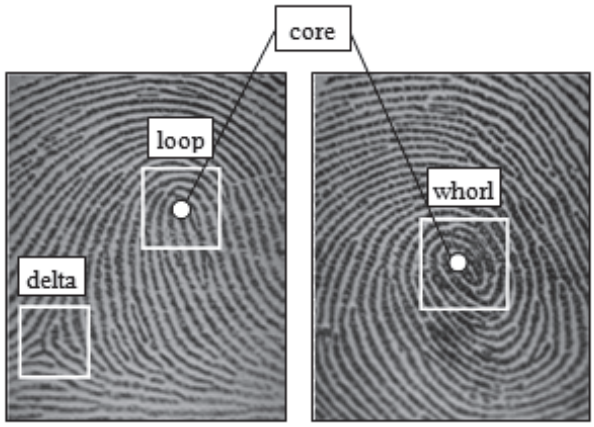


Рисунок 1.2 – Особые точки.

Узоры на коже пальцев рук образуются папиллярными линиями. Почти параллельные между собой, они образуют на поверхности кожи общие признаки – узоры в виде петель, дельт, завитков и карманов, сочетание которых определяет

тип узора. Линии оканчиваются или начинаются, соединяются или расщепляются, образуя частные признаки.[14]

Выделяют три основные группы классификации узора: дуговые, петлевые и завитковые – рисунок 1.3.

Рисунок 1.3 – Отпечатки пальцев, относящиеся к трем различным группам (слева направо): дуговые, петлевые и завитковые.

При этом дуговые и петлевые могут быть также быть подразделены на подгруппы. С петлями, ориентированными влево и вправо. Просто дуговые и шатровые – рисунок 1.4.

Рисунок 1.4 – Классификация с подгруппами

Отпечатки пальцев разных людей могут иметь одинаковые глобальные признаки, но считается невозможным наличие одинаковых деталей (локальных признаков). [15] Поэтому глобальные признаки используют для разделения базы данных на классы и на этапе аутентификации. На втором этапе распознавания (идентификация) используют уже локальные признаки.

Верификация или аутентификация производит попарное сравнение имеющегося и предоставленного набора информации, с их помощью реализуется подтверждение личности человека. Результатом является указание доли вероятности опознания человека.

В ходе же идентификация один и тот же входной отпечаток сравнивается со всеми имеющимися в базе данных. Из-за ее большого размера вводится классификация и, таким образом, входной отпечаток сравнивается только с имеющимися отпечатками этого же класса.

Но учитывая, что получить идеальный отпечаток пальца в реальных условиях почти невозможно, они будут загрязнены, деформированы, смазаны, палец по поверхности может проскальзывать, иметь шрамы или ожоги, помимо этого следует учитывать факторы, связанные со свойствами следообразующего материала, с деформацией пальца в момент следообразования. Все вышеперечисленное приведет к искажению модели отпечатка пальца и снижению точности и надежности идентификации папиллярного узора.

Для преодоления этой проблемы были разработаны методы фильтрации изображений.

В едином процессе идентификации по дактилоскопическому узору можно выделить этапы: [16]

1) получение изображения с сенсора, вероятно с дефектами;
2) предварительная обработка, на выходе которой будет очищенный от помех скелет отпечатка пальца, с отмеченными особыми точками;
3) непосредственно сравнение.

Данная работа сконцентрирована на втором этапе. Здесь изображение отпечатка, полученное от сенсора, перед началом процесса идентификации и сопоставления должно пройти этап предварительной обработки. Она заключается в применении к исходному изображению оператора выделения линий узора, из которых, в дальнейшем, будет получено бинарное (чёрно-белое) изображение, пригодное для дальнейшей обработки (скелетизации, выделения особых точек).

Необходимо избавиться от помех, возникших в процессе считывания изображения от сенсора, выделить на отпечатке информативные области и обозначить цветом, непосредственно линии узора – бинарное изображение.

Бинарное изображение отпечатка пальца представляет собой булеву матрицу размера изображения, поступившего от считывающего отпечаток пальца устройства, каждый нулевой элемент которой соответствует области гребней, а каждый единичный – области бороздок изображения папиллярного узора. [16]

1.3 Формулировка задания

Необходимо разработать и реализовать на языке программирования алгоритм, который из входного изображения отпечатка пальца, например, полученного с какого-либо сенсора, выделит бинарный узор пригодный для построения скелета и дальнейшего анализа.

Фильтрация изображения должна быть основана на алгоритме фильтра Габора.

Исходя из того, что для выполнения фильтрации данным методом необходима следующая информация: [17]

1) данные о информативной области изображения;
2) данные о плотности линий изображения;
3) данные о направлении линий изображения.

Следует также разработать и реализовать алгоритмы для нахождения этой информации.

2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

2.1 Функциональные требования

Необходимо реализовать фильтрацию и построение бинаризированного изображения на основе исходного для последующей скелетизации.

При этом необходимость обрабатывать различные краевые эффекты при применении методов, а также эффекты на границе изображения отсутствует.

Как подготовительные этапы перед фильтрацией необходимо реализовать нахождение информативной области изображения, поля потоков и поля направлений.

Результаты применения методов каждого этапа обработки изображения должны быть наглядными. Таким образом, на экран должны выводиться исходное изображение и обработанное. Должна присутствовать возможность выбора этапа в любой момент.

Необходимо реализовать возможность построения гистограммы и сглаживания изображения произвольной маской 3x3.

Среди обязательных этапов обработки изображения должны присутствовать:

1) построение модуля градиента;
2) построение направления градиента;
3) построение модуля градиента в окрестности;
4) построение направления градиента в окрестности;
5) построение когерентности изображения;
6) построение поля плотности изображения;
7) построение поля направления изображения;
8) визуализация поля направлений линиями;
9) минимум два различных показателя качества выполнения построения поля плотности;
10) визуализация информативной области;

11) результат работы фильтра Габора (бинаризированный);

12) скелет изображения;

13) визуализация особых точек на скелете (разветвления и окончания).

2.2 Требования к ПО

Программа для демонстрации работы реализованных алгоритмов должна быть написана на языке C#.

Программа должна быть оформлена в виде графического приложения на технологии Windows Forms.

Демонстрируемые изображения должны отображаться в контейнере PictureBox.

Всё время работы программы должно отображаться исходное изображение.

Должна быть предусмотрена возможность сглаживания изображения произвольной матрицей 3x3 и построение гистограммы.

Для гистограммы должна быть отдельная область в виде графика.

Все этапы обработки изображения должны отображаться рядом с исходным изображением и быть доступны в любой момент с помощью выпадающего списка.

Уже выбранные ранее этапы не должны пересчитываться при повторном выборе.

Должна быть предусмотрена кнопка полного сброса.

Если для построения изображений на каких-либо этапах используются числовые параметры, должна быть возможность задать их в программе.

Кроме того, должна быть возможность получения координат положения курсора мыши на обработанном изображении.

3 ПРЕДВАРИТЕЛЬНЫЙ ЭТАП

Перед началом разработки приложения построим план работы, конечной целью работы которого является выделение особых точек на изображении. Перед этим исходное изображение должно пройти несколько этапов обработки, чтобы стать пригодным к применению данной операции. В результате чего был проведен анализ методов, которые нужно использовать и их последовательность для получения конечного результата.

Разобьем разработку приложения по обработке изображения отпечатка пальца на два основных этапа – это фильтрация и выделение особых точек.

Чтобы отфильтровать изображения нужно получить информативную область и поле плотности. Для построения поля плотности требуется построить модуль градиента изображения и его направление. Для получения информативной области будем использовать несколько параметров, в том числе и комплексно. Соответственно на данном этапе будем использовать метод для построения когерентности, а также поля плотности.

После применения фильтрации и перед процессом выделения особых точек, требуется построить скелет изображения.

Общий план работы представлен на рисунке 3.1.

Исходное изображение, на котором будет проводиться проверка работы программы, представлено на рисунке 3.2. Данное изображение в целом хорошего качества, поэтому будет видно изъяны используемых методов, что поможет при их исследовании.

Кроме того, перед выполнением основных этапов можно произвести предварительное сглаживание изображения, так как это улучшает качество выполнения последующих шагов.

При этом входное изображение может быть и цветным, однако в данной работе производится работа только с одним слоем – яркостью. Преобразование цветного изображения к черно-белому с параметром яркости пикселя

изменяющемся в пределах от нуля до двухсот пятидесяти пяти происходит с помощью стандартных методов языка C#. [23]

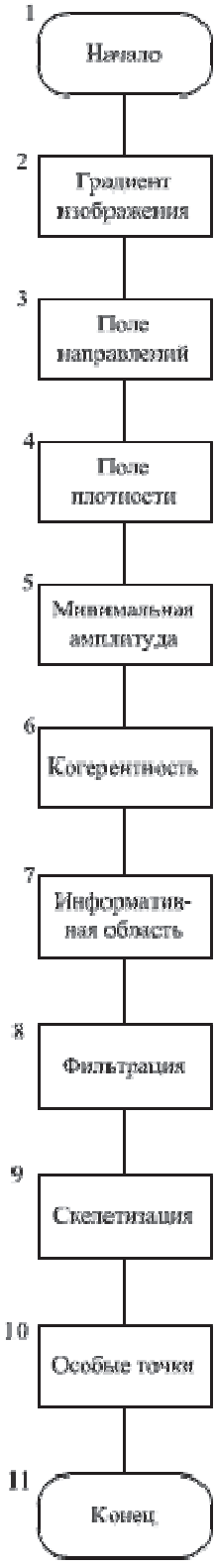


Рисунок 3.1 – Общая блок – схема

Рисунок 3.2 – Исходное изображение

4. ОСНОВАНАЯ ЧАСТЬ

Чтобы построить качественный скелет изображения и выделить особые точки в данной работе оно будет обработано фильтром Габора, а для этого требуются сведения, описанные в разделе 1.4:

1) поле направления изображения;
2) поле плотности изображения;
3) информативная область.

4.1 Построение поля направлений

4.1.1 Сглаживание

В качестве способа сглаживания был выбран метод обработки изображений матрицей свертки.

Суть алгоритма заключается в следующем: [18]

К каждому пикселю исходного изображения применяется так называемая матрица свертки.

Коэффициенты этой матрицы умножаются на яркости пикселя изображения под маской, суммируются и делятся на коэффициент нормирования, для того, чтобы общая яркость изображения осталась неизменной.

Коэффициенты матрицы свертки могут варьироваться, самые распространённые представлены ниже:

$$\begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} \quad \text{и} \quad \begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix} \tag{4.1}$$

Применение матрицы свертки к изображению показано формулой 4.2.

$$\begin{vmatrix} b9 & b2 & b3 \\ b8 & b1 & b4 \\ b7 & b6 & b5 \end{vmatrix} * \begin{vmatrix} k9 & k2 & k3 \\ k8 & k1 & k4 \\ k7 & k6 & k5 \end{vmatrix} * \frac{1}{div} = \begin{vmatrix} b9 * k9 & b2 * k2 & b3 * k3 \\ b8 * k8 & b1 * k1 & b4 * k4 \\ b7 * k7 & b6 * k6 & b5 * k5 \end{vmatrix} * \frac{1}{div} \tag{4.2}$$

b1…b9 – яркости входного изображения.

k1…k9 – коэффициенты матрицы.

div – коэффициент нормирования, для того чтобы средняя интенсивность изображения оставалась неизменной.

Но следует учесть и граничные условиях, характерные для всех матричных фильтров. У граничных пикселей не существует соседа, поэтому не на что умножать коэффициенты матрицы. [18] Таким образом, можно выйти за границы массива. Данная проблема иллюстрируется на рисунке 4.1.

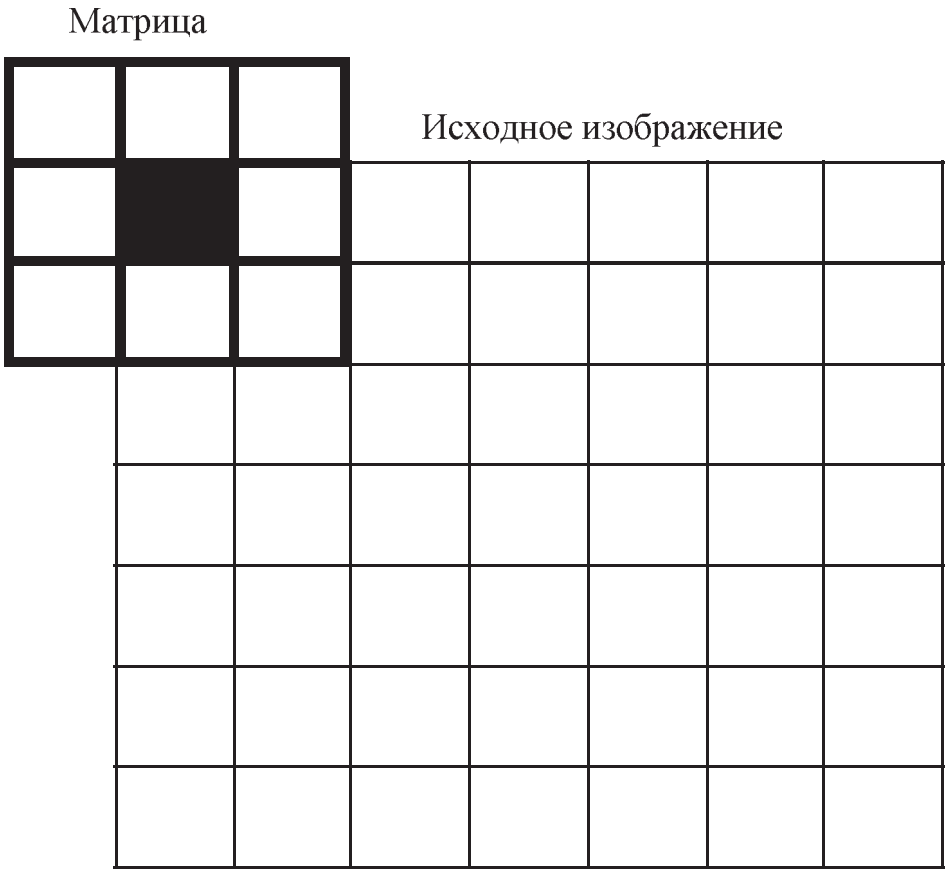


Рисунок 4.1 – Выход применяемой матрицы за границы изображения

Для решения возникшей проблемы будем использовать рабочую область, внутри исходного изображения путем вычитания по одному пикселю с краев – рисунок 4.2.

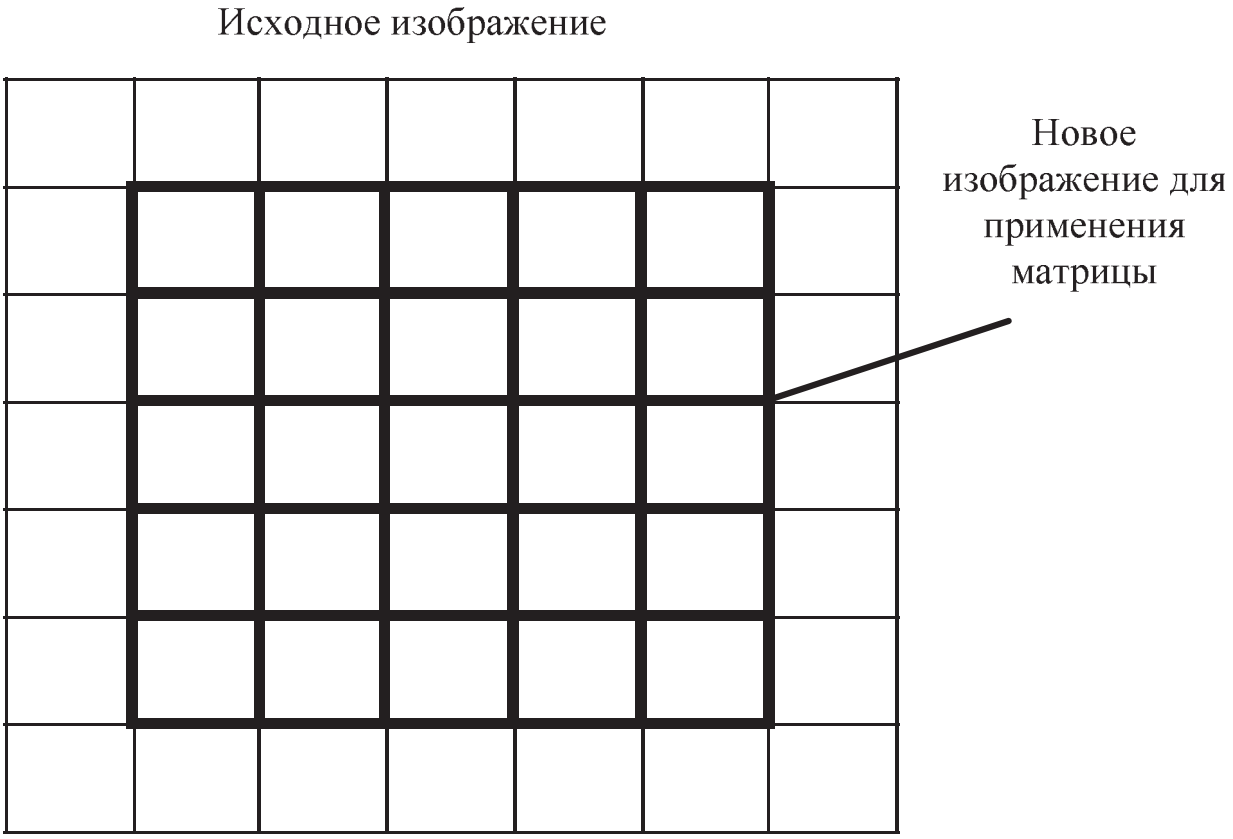


Рисунок 4.2 – Рабочая область изображения для сглаживания

Демонстрация применения представленного метода с использованием различных матриц – формула 4.1– демонстрируется на рисунке 4.3.

а) б) в)

Рисунок 4.3 – Сглаживание изображений: исходное изображение(а), взвешенное размытие(б), равномерное размытие(в)

Среди достоинств данного метода можно отметить простоту реализации, быструю скорость работы и хороший результат сглаживания. В качестве недостатка выступают граничные необработанные пиксели. Было принято пренебречь данным фактом, так как техническое задание не оговаривает это условие. В дальнейшем будем использовать равномерное размытие.

4.1.2 Градиент в точке

Градиент – вектор, показывающая направление наискорейшего возрастания некоторой величины. В нашем случае «некоторая величина» – это двумерная функция яркости изображения. [19]

По определению, градиент изображения $f(x, y)$ в точке (x, y) – это вектор [19]:

$$\nabla f = \begin{bmatrix} Gx \\ Gy \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}, \quad (4.3)$$

где $\frac{dI(x,y)}{dx}$ – изменения яркости соседних пикселов по оси X;

$\frac{dI(x,y)}{dy}$ – изменения яркости соседних пикселов по оси Y.

Как известно из курса математического анализа, направление вектора градиента совпадает с направлением максимальной скорости изменения функции в точке [19].

Вектор градиента, как и любой другой вектор имеет две основные характеристики – это модуль и направление.

Модуль вектора обозначается и равен:

$$|\nabla f| = \sqrt{\mathrm{G_x^2} + \mathrm{G_y^2}} \quad (4.4)$$

Эта величина равна значению максимальной скорости изменения функции f в точке (x, y), причем максимум достигается в направлении вектора ∇f. Величину ∇f также часто называют градиентом. [20]

Направление вектора градиента является углом $\alpha(x, y)$ между направлением вектора ∇f в точке (x, y) и осью x. [19]

$$\alpha(x, y) = arctg\left(\frac{G_y}{G_x}\right) \quad (4.5)$$

Отсюда легко найти направление контура в точке (x, y), которое перпендикулярно направлению вектора градиента в этой точке. А вычислить градиент изображения можно, вычислив величины частных производных $\frac{\partial f}{\partial x}$ и $\frac{\partial f}{\partial y}$ для каждой точки.

На практике, картинка дискретна и в классическом смысле производной не существует, поэтому используют другие подходы, позволяющие найти приближенные значения производных. В данной работе для этой цели будет использоваться оператор Собеля [20].

Оператор применяется к изображению в виде матрицы свертки также, как описано в предыдущем разделе и обладает аналогичными достоинствами и недостатками.

Сам оператор представляет собой следующие матрицы:

$$\begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix} \quad \text{и} \quad \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix} \quad (4.6)$$

Очевидно, что первая матрица дает наибольший отклик у вертикальных границ, а вторая у горизонтальных, что дает приближенное значение производных $\frac{\partial f}{\partial x}$ и $\frac{\partial f}{\partial y}$.

Рассчитав в каждой точке значение по первой и второй матрице можно вычислить модуль и направление градиента по формулам 4.4 и 4.5.

Результат применения описанного способа представлен на рисунке 4.4.

Стоит отметить, что в данной работе была реализована функция целочисленного арктангенса в диапазоне от нуля до двухсот пятидесяти пяти градусов. Она заменила собой стандартную функцию языка C#. Точности такой функции достаточно для целей данной работы, а скорость заметно возрастает.

Исходный код функции в Приложении А.

а) б)

Рисунок 4.4 – Модуль градиента(а) и его направление(б)

4.1.3 Градиент в области

Для построения поля направлений можно воспользоваться методом нахождения модуля и направления градиента в некоторой окрестности.

Предположим, мы будем искать модуль и направления векторов градиента в окрестности S. Размеры окрестности $I * J$ при $(I = J)$.

Обозначим значение модуля градиента в точке (i, j) $|\nabla f|\,(i, j)$, а направление $\alpha(i, j)$.

Тогда искомые значения в точке (x, y) изображения – $|\nabla F|\,(x, y)$ и направление $A(x, y)$.

$$X = \sum_{i=1}^{I}\sum_{j=1}^{J}|\nabla f|\,(i,j) * \cos\,(2 * \alpha(i,j)) \tag{4.6}$$

$$Y = \sum_{i=1}^{x}\sum_{j=1}^{y}|\nabla f|\,(i,j) * sin\,(2 * \alpha(i,j)) \tag{4.7}$$

$$|\nabla F|\,(x,y) = \sqrt{X * X + Y * Y} \tag{4.8}$$

$$\mathrm{A}\,(x,y) = Atan(\frac{Y}{X})/2 \tag{4.9}$$

Для дальнейшей обработки был выбран размер окрестности S 15x15, так как в ходе опытов это оказался наиболее оптимальный вариант. Результат расчетов по формулам 4.8 и 4.9 представлен на рисунке 4.5.

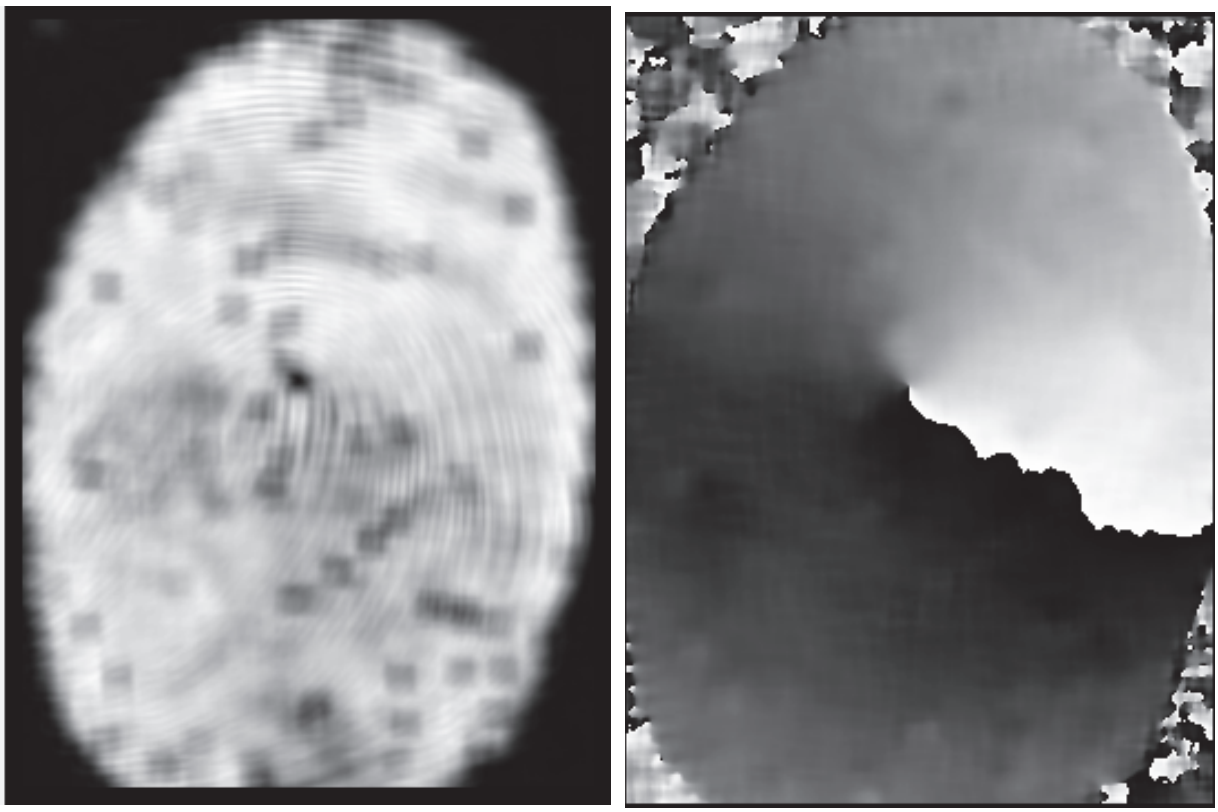


Рисунок 4.5 – Модуль и направление градиента в окрестности (апертура 15)

Для сравнения приведем изображения с меньшим размером окрестности S – рисунок 4.6. Больший размер делать не имеет смысла, так как объем вычислений возрастает, а качество уже на данном этапе является приемлемым.

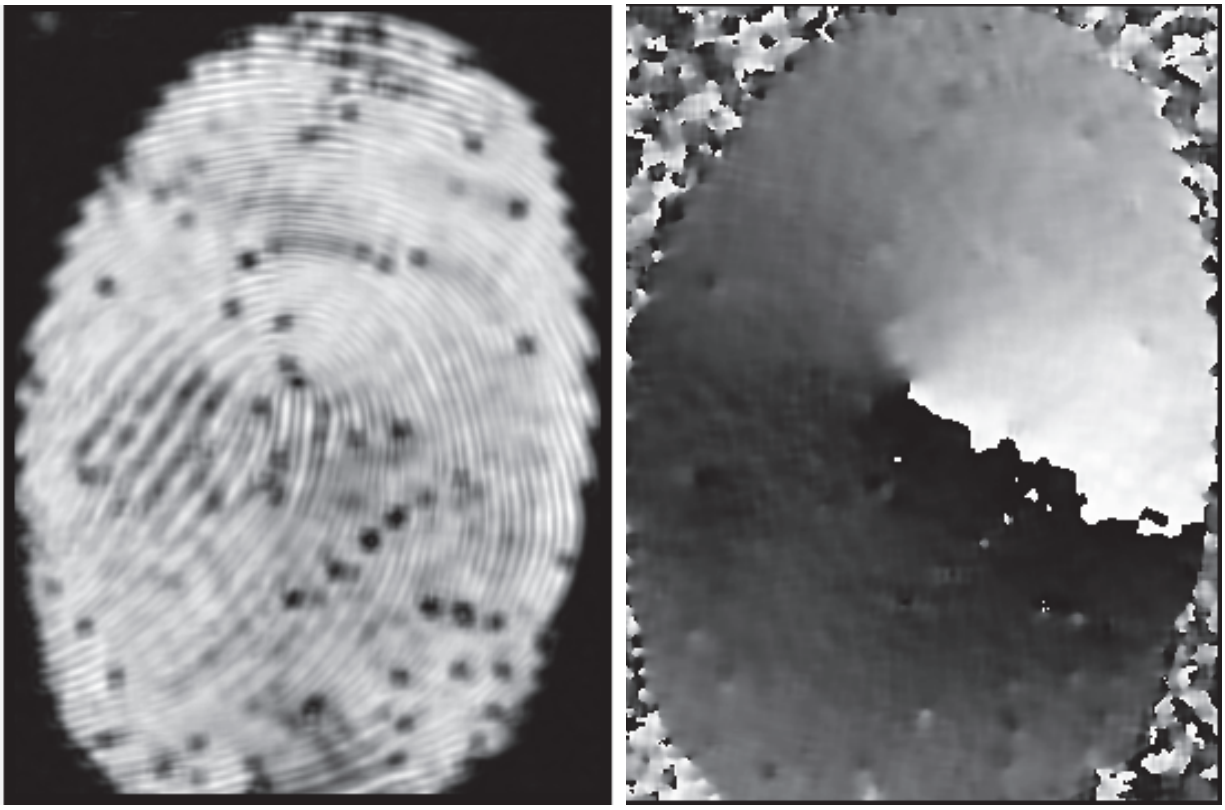


Рисунок 4.6 – Модуль и направление градиента в окрестности (апертура 9)

Как видно по рисунку 4.6 при уменьшении апертуры появляются артефакты и изображение становится менее плавным, особенно это заметно на направлении градиента.

Реализованный метод основан на алгоритме, предложенным M.Bazen – формулы 4.8, 4.9 [21, 21с].

Однако у Базена модули складываются квадраты модулей, что увеличивает чувствительность к выбросам длин векторов. В данной же работе идет сложение оригинальных модулей, что увеличивает сглаженность.

4.1.4 Поле направлений

Основываясь на данных предыдущего пункта и зная, что направление линий узора перпендикулярно направлению градиента в области можно построить так называемое поле направлений.

Для визуализации поля направлений покажем направление в каждой выбранной окрестности линиями.

Для построения черточек воспользуемся алгоритмом Брезенхема, разработанным Джеком Элтоном Брезенхэмом в 1962 году.

Алгоритм Брезенхема – это алгоритм, определяющий, какие точки двумерного растра нужно закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками. [7]

Состоит он в следующем:

Отрезок проводится между двумя точками – (x_0, y_0) и (x_1, y_1), где в этих парах указаны столбец и строка соответственно, номера которых растут вправо и вниз. Сначала будем предполагать, что линия идёт вправо и вниз, причём горизонтальное расстояние $x_1 - x_0$ превосходит вертикальное $y_1 - y_0$, то есть наклон линии от горизонтали – менее 45°. Цель состоит в том, чтобы для каждого столбца x между x_0 и x_1 определить, какая строка y ближе всего к линии, и нарисовать точку (x, y).

Общая формула линии между двумя точками:

$$y - y_0 = \frac{y_1 - y_0}{x_1 - x_0} * (x - x_0) \qquad (4.10)$$

Поскольку мы знаем колонку x, то строка y получается округлением к целому следующего значения:

$$y = \frac{y_1 - y_0}{x_1 - x_0} * (x - x_0) + y_0 \qquad (4.11)$$

Однако, вычислять точное значение этого выражения нет необходимости. Достаточно заметить, что y уменьшается от y_0 и за каждый шаг мы добавляем к x единицу и добавляем к y значение наклона (в нашем случае значение наклона будет отрицательным числом):

$$S = \frac{y_1 - y_0}{x_1 - x_0} \quad (4.12)$$

которое можно вычислить заранее. Более того, на каждом шаге мы делаем одно из двух: либо сохраняем тот же y, либо уменьшаем его на 1.

Что из этих двух выбрать – можно решить, отслеживая значение ошибки, которое означает – вертикальное расстояние между текущим значением y и точным значением y для текущего x. Всякий раз, когда мы увеличиваем x, мы увеличиваем значение ошибки на величину наклона s, приведённую выше в формуле 4.12. Если ошибка превысила 0.5, линия стала ближе к следующему y, поэтому мы уменьшаем y на единицу, одновременно уменьшая значение ошибки на 1. Далее применяем этот алгоритм во всех направлениях.

Но, так как мы имеем данные только о направлении линии и координатах её центра, необходимо получить координаты начала и конца линии для работы алгоритма Брезенхема.

Для этого можно воспользоваться следующими формулами:

$$\begin{gathered} startX = x \\ startY = y \\ endX = x + line * Sin(angle) \\ endY = y + line * Cos(angle) \end{gathered} \quad (4.13)$$

Где line – длина линии. [22]

Для целей данной работы был выбран именно алгоритм Брезенхема так как он хорошо описан в литературе, достаточно быстро работает, и главное строит линию толщиной в один пиксель, что является необходимым требованием для последующих этапов.

Результат визуализации поля направлений алгоритмом Брезенхема представлен на рисунке

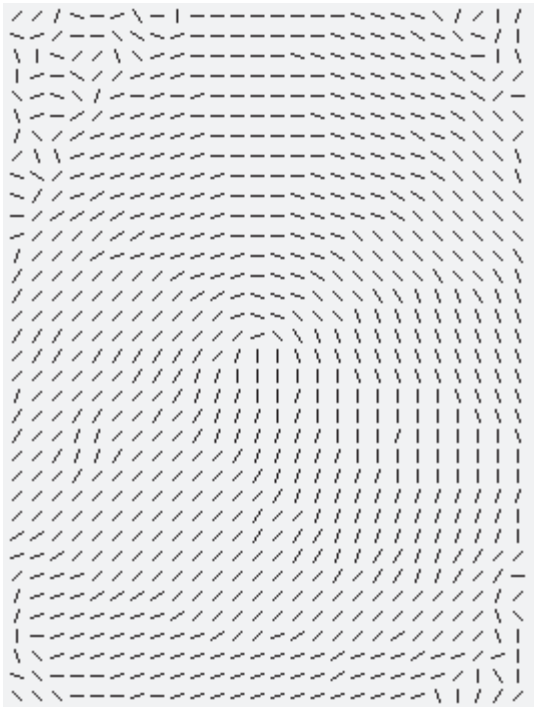


Рисунок 4.7 – Визуализация поля направлений линиями

4.2 Построение поля плотности

Частота линий варьируется на протяжении всего отпечатка пальца. Поле плотности можно представить, как частоту линий папиллярного узора в окрестности точки. Для нахождения плотности в литературе представлено несколько методов.

Hong, Wan, и Jain вычисляют локальную частоту линий путем подсчета среднего значения по количеству пикселей между двумя последовательными вершинами серого уровня вдоль направления нормали к локальным направлениям линий.

Jiang вычислял локальную частоту линий, начиная с x – сигнатуры. Он использовал технику спектра высокого порядка, называющегося mix-спектр.

Поход Maio и Maltoni (1998a) основывался на шаблоне линии, которую локально моделируют как синусоидальную форму поверхности и, далее используется теорема вариации для оценки неизвестной частоты линий. [1, с 112]

В статье «Sex determination from fingerprint ridge density» приводятся следующий метод: на пальце каждого человека была выбрана зона размером 5x5

мм, в которой по направлению диагонали был произведен подсчет линий. Были учтены некоторые специфические критерии, например, точки не были подсчитаны, а некоторые другие паттерны считались как две линии. В результате исследования была установлена граница в 13 – 14 линий на 25 квадратных миллиметров. Вычисления производились на всех десяти пальцах, после чего вычислялось среднее значение. Если частота меньше этой границы вероятно индивидуум является мужчиной, если больше вероятно женщиной. [8]

Для нахождения плотности мы воспользуемся формулой, предложенной Д. Мальтони [23], однако координаты вершин найдем по собственному алгоритму.

Идея алгоритма заключается в следующем:

Для каждой точки изображения по информации о направлении градиента в области, найденной в разделе 4.1.3. мы можем найти и локальное направление линий узора, которое перпендикулярно направлению градиента.

Имея данную информацию, воспользовавшись алгоритмом Брезенхема, описанном в разделе 4.1.4. можно построить линию определенной длины перпендикулярно направлению линий узора – рисунок 4.8, красные точки – центр и окончания проведенной линии.

После этого необходимо последовательно считать значения градиента в каждом пикселе изображения в некий буфер B (рисунок 4.9). Номер позиции в буфере обозначим x.

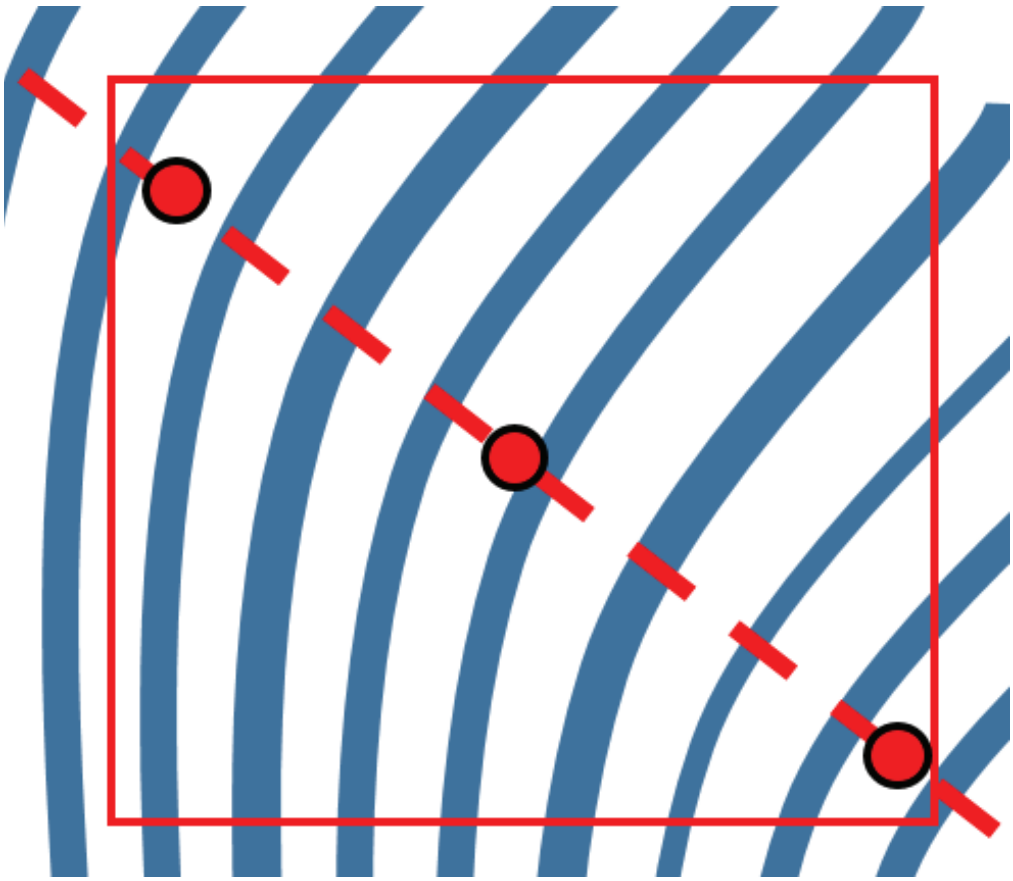


Рисунок 4.8 – Линия, перпендикулярная узору

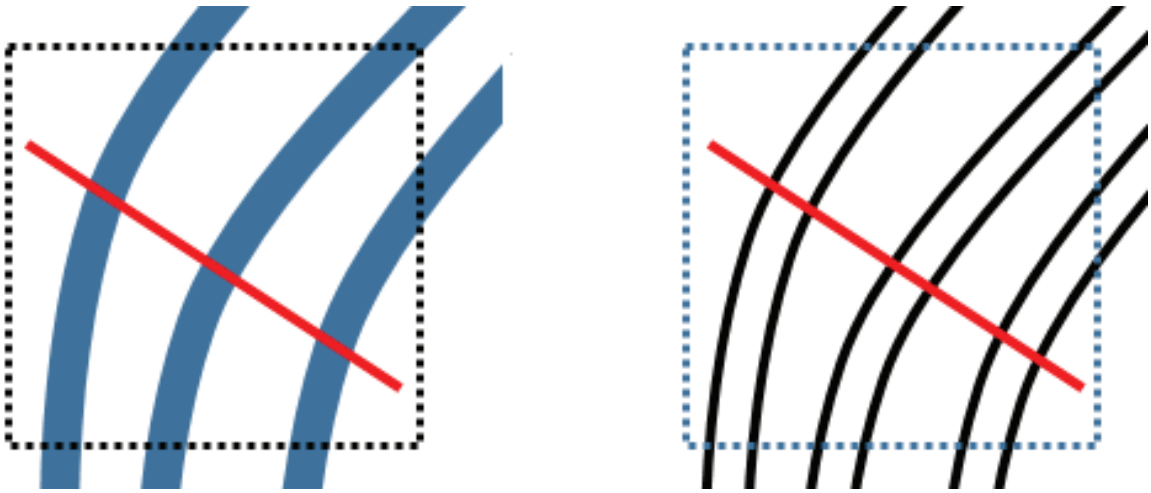


Рисунок 4.9 – Оригинальные линии узора и градиент

Теперь можно сказать следующее – в созданном буфере есть последовательные значения функции градиента, в направлении перпендикулярном узору и в центре с исследуемой точкой изображения:

$$f(x) = grad(x), \tag{4.14}$$

где $x -$ координата в буфере.

Исходя из задачи – подсчета линий на узоре можно было бы сразу сказать, что на границе линии будет наибольшее значение градиента. Однако не всякое наибольшее значение будет соответствовать именно линиям узора, а может быть и результатом помех.

Поэтому можно воспользоваться следующим – зная, что производная показывает скорость изменения функции в точке, и при этом если функция возрастает, то производная положительная, а если убывает, то отрицательная, можно посчитать производную в каждой из сохраненных точек функции $grad'(x)$.

Для подсчета искомых значений можно воспользоваться определением производной в точке:

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \tag{4.15}$$

Тогда, в рассматриваемом случае:

$$grad'(x_0) = grad(x_0 + 1) - grad(x_0) \tag{4.16}$$

Выполнив расчет по указанной формуле и соответственно записав результаты расчетов в буфер аналогичного размера B', можно заметить, что записанные значения напоминают значения периодической функции $Cos(x)$ или $Sin(x)$. Где точки экстремума (максимума) приходятся на границах линий – рисунок 4.10. Значение в точке максимума будем называть амплитудой.

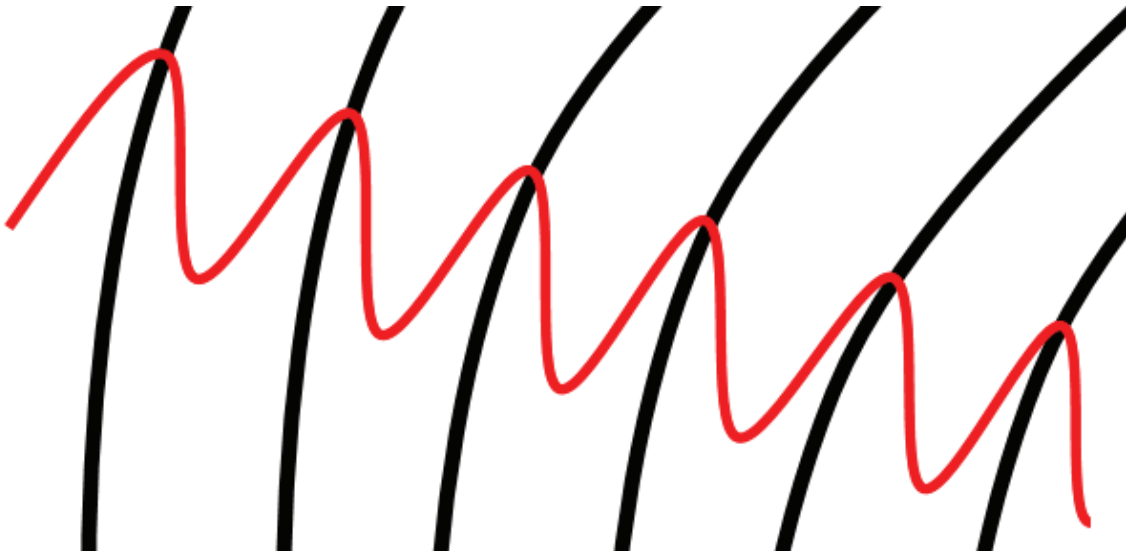


Рисунок 4.10 – Производная по градиенту

Значения функции в буфере B' пригодны для решения задачи нахождения плотности линий. Так, по данным значениям можно найти точки максимума и, имея их координаты для нахождения расстояний, воспользоваться формулой 4.10.

$$f = \frac{n}{S_1 + S_2 + \cdots + S_n} \tag{4.17}$$

В формуле 4.17 n – число расстояний между точками максимума, а S_n – соответственно, расстояния между соседними точками.

При этом можно сделать дополнительное усовершенствование алгоритма – учитывать не все точки максимума, а лишь те, что выше некоторого порога, который необходимо подбирать для каждого изображения индивидуально в зависимости от интенсивности изображения. Это решение даст избавление от помех, вызванных незначительными перепадами на фоне или между и/или на линиях. Иллюстрации изменения порога приведены на рисунках 4.12 и 4.13.

Таким образом получается отследить лишь наиболее значительные изменения градиента, которые соответствуют границам линий.

Результат работы алгоритма можно видеть на рисунке 4.11.

Светлые участки обозначают высокую плотность линий, темные – соответственно наоборот.

Среди достоинств используемого метода в данной работе отметим его скорость работы, она достигается за счёт анализа линии, а не области, как в

некоторых из предложенных методов, кроме того используется приближенная производная градиента, что дает четкие отклики на границах линии и позволяет избежать помех на фоне.

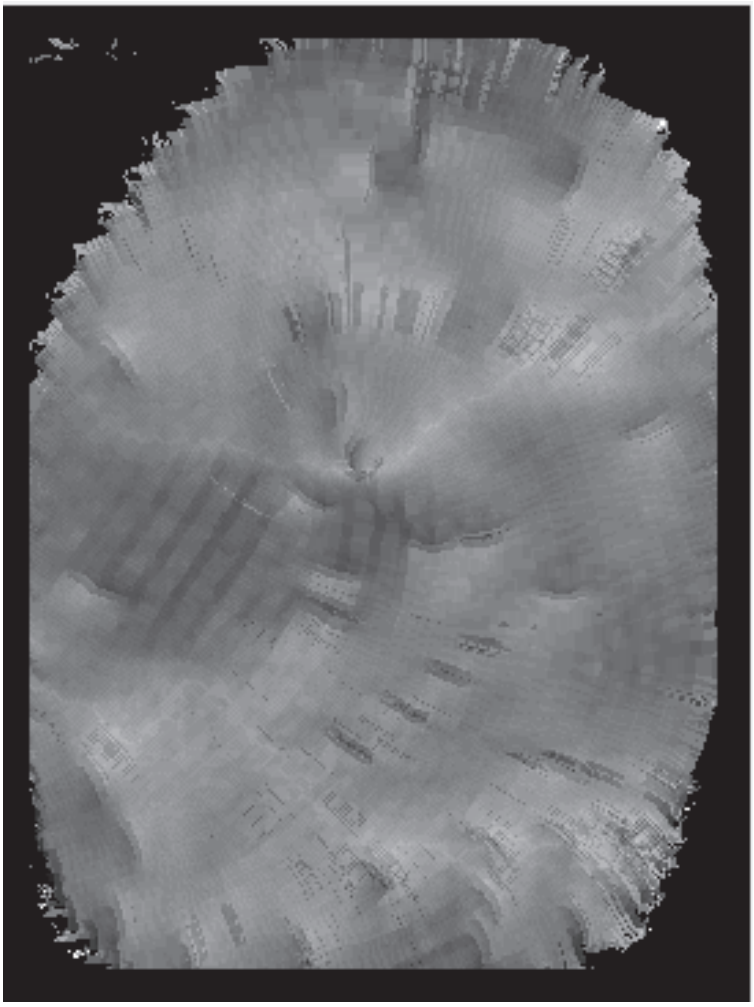


Рисунок 4.11 – Плотность линий (средняя)

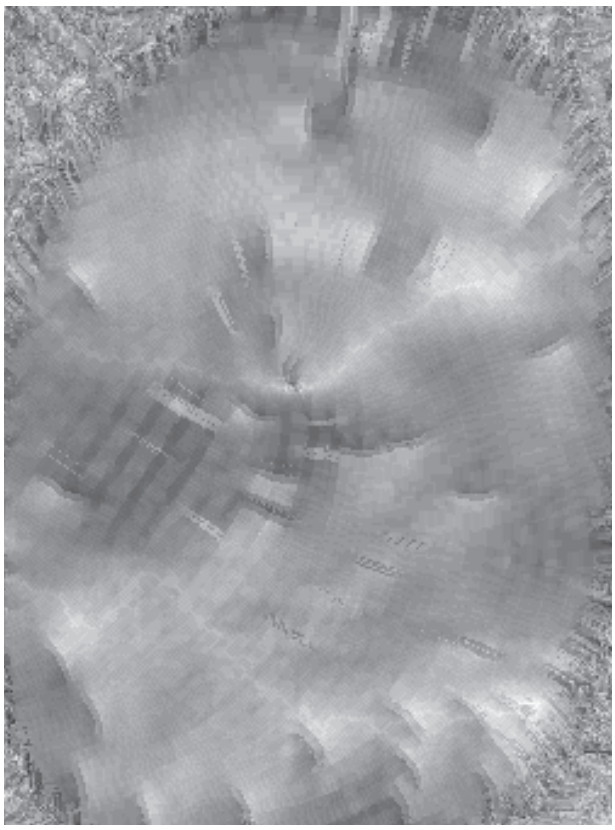


Рисунок 4.12 – Плотность линий без учета порога

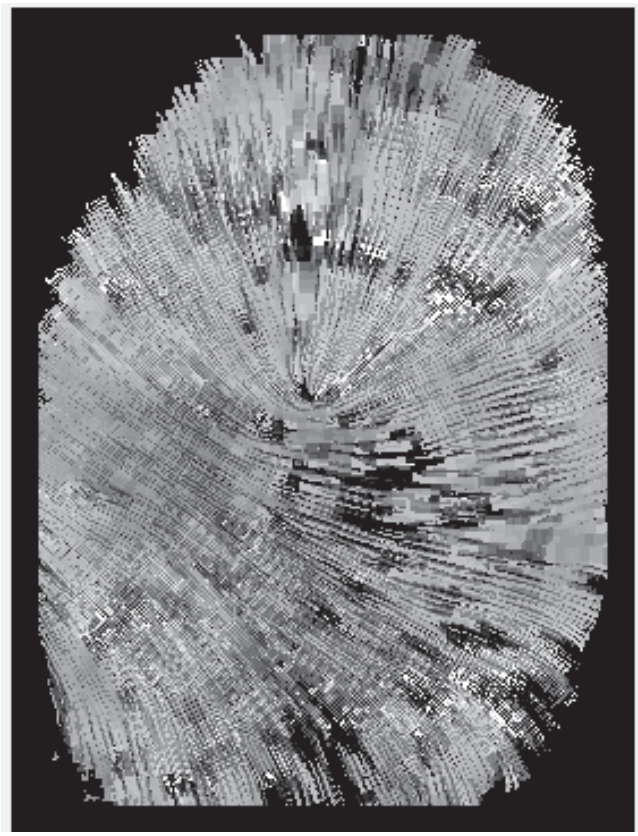


Рисунок 4.13 – Плотность линий с высоким порогом

4.3 Построение информативной области

Последними недостающими данными перед фильтрацией являются координаты пикселей, составляющих информативную область.

При фильтрации обрабатывается все изображение, хотя не все фрагменты относятся к отпечатку пальца. Что приводит к различного рода помехам на выходном изображении. Для того чтобы выделить области, относящиеся к изображениям линий, применяются различные подходы.

В частности, является допустимым подход, при котором информативная область сразу строится по полю плотности изображения. Однако, тут присутствуют свои недостатки, например, это рваные края самой области, возможные пустоты внутри области или островки за её пределами.

Чтобы минимизировать данные отклонения в данной работе будут использоваться показатели качества построения поля плотности, такие как когерентность, «Волосы вероники» и комплексный показатель, учитывающий два предыдущих.

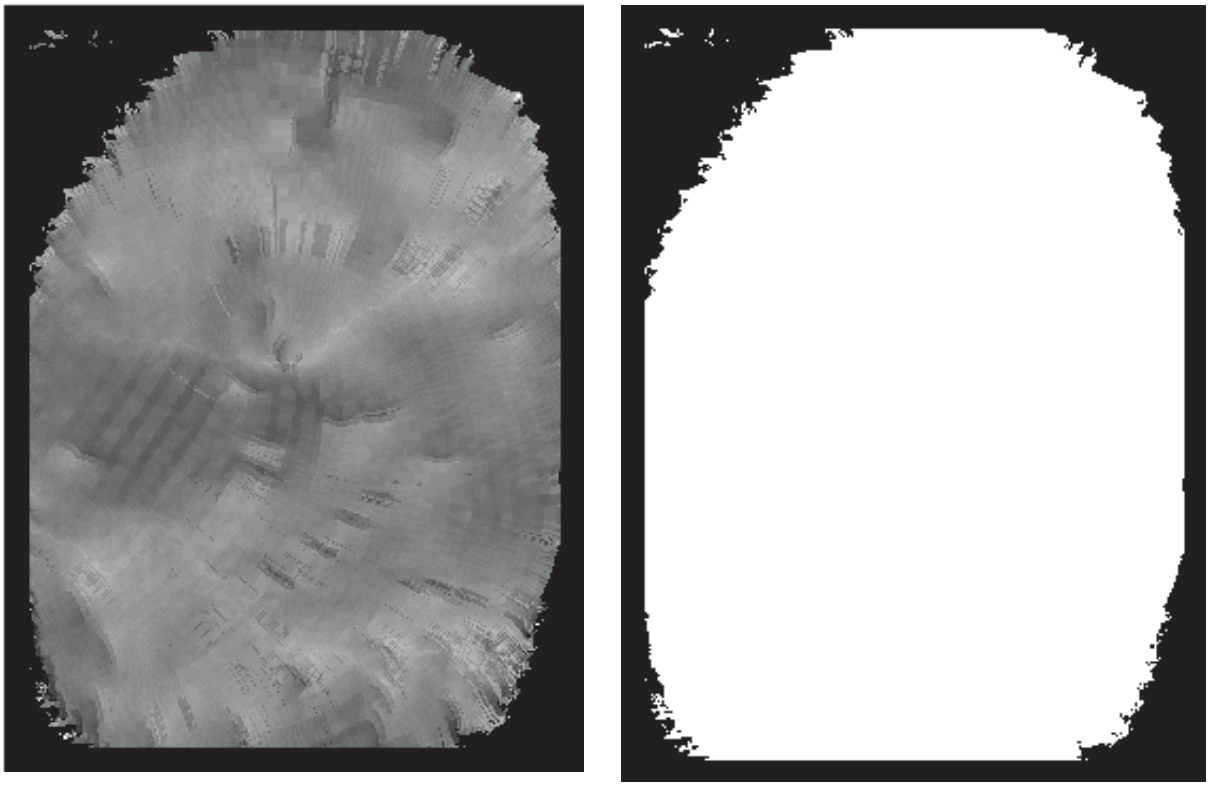
Для демонстрации необходимости проведения последующих этапов приведено поле плотности изображения и построенная по нему информативная область.

Как видно по рисунку 4.14 на данном этапе еще преждевременно проводить фильтрацию.

Для небольшого улучшения качества нахождения информативной области можно сгладить исходное изображение.

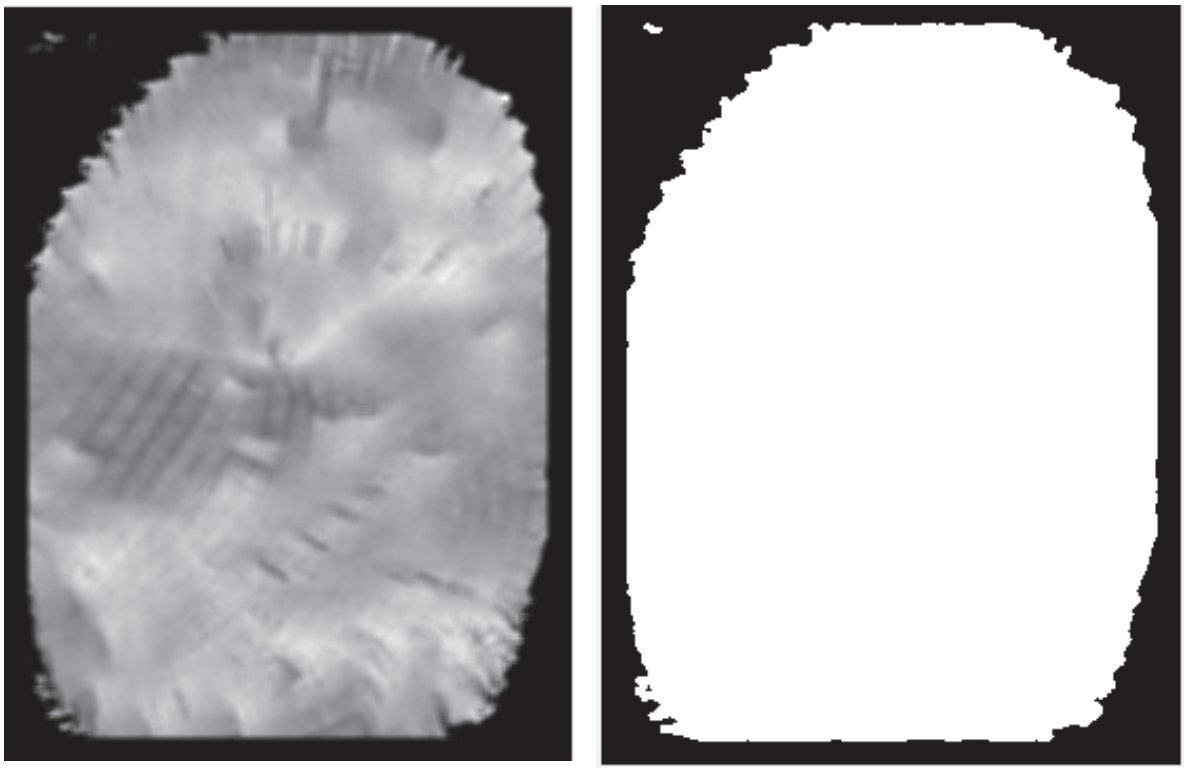
Сглаживание немного улучшает краевые эффекты, но кардинально не меняет ситуацию.

Рисунки 4.14 (б), 4.15 (б), представляющие информативную область получены путем бинаризации изображения по гистограмме (рисунок 4.16).



а) б)

Рисунок 4.14 – Поле плотности (а), его бинаризация (б)



а) б)

Рисунок 4.15 – Сглаженное поле плотности (а), его бинаризация (б)

Идея такой бинаризации заключается в следующем – по исходному изображению строится гистограмма, по ней подсчитывается максимум. От него отбрасывается одна треть и, берется квантиль, например, 30%, который и разделит всё изображение на узор и фон.

Это общее представление метода. Чтобы получить наилучший результат были произведены эксперименты по изменению квантиля. На рисунках 4.17 и 4.18 представлены результаты исследования.

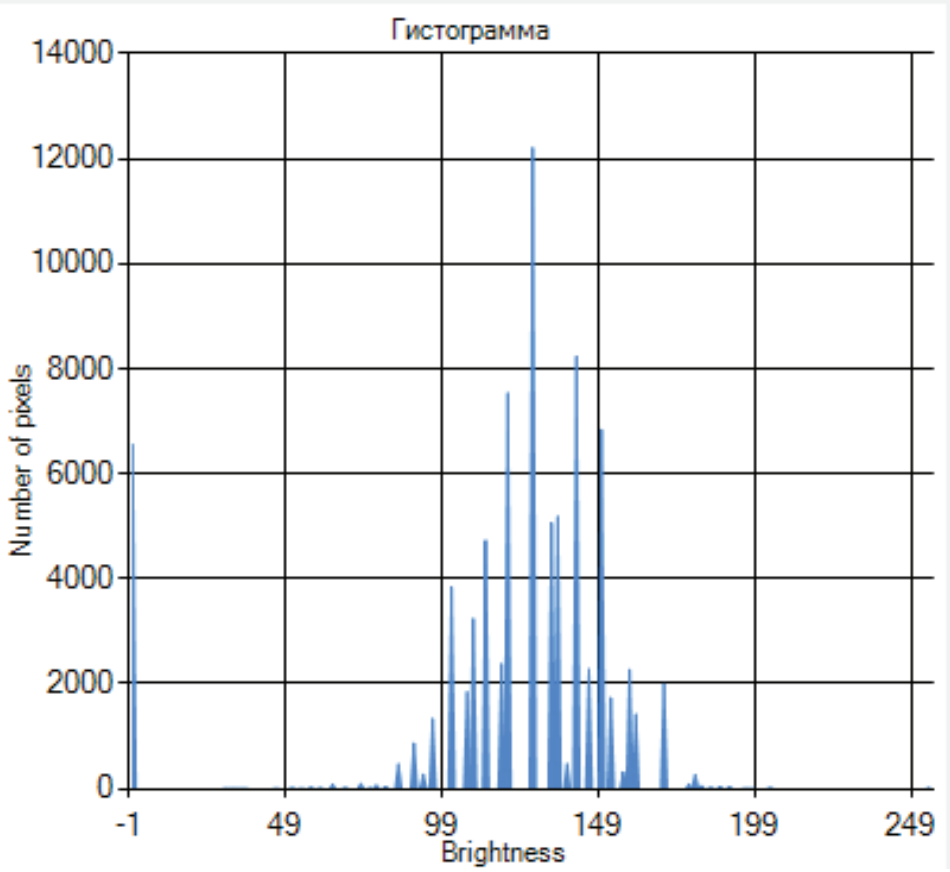


Рисунок 4.16 – Гистограмма рисунка 4.14(а)

Рисунок 4.17 – Низкое значение квантиля

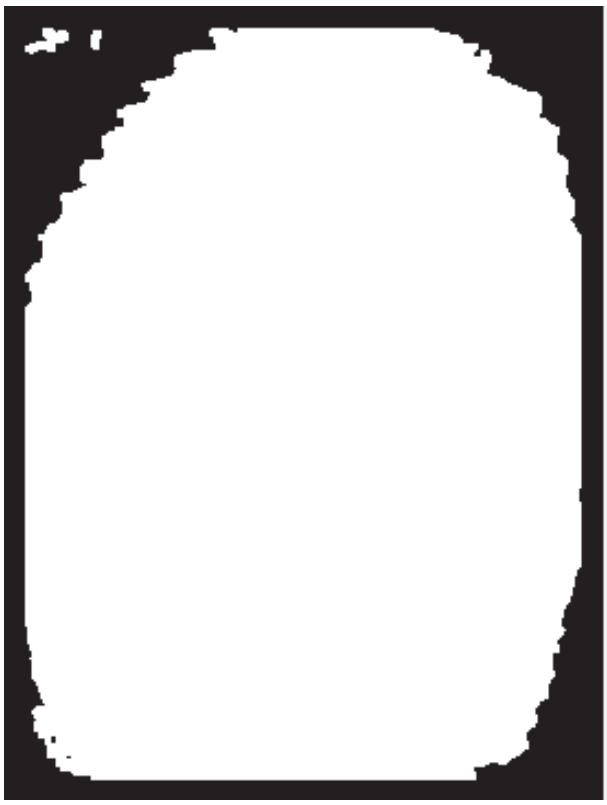


Рисунок 4.18 – Высокое значение квантиля

4.3.1 Показатели качества

4.3.1.1 Построение когерентности

Когерентность применяется для того чтобы выделить области, относящиеся к изображениям линий отпечатка, рассчитывается когерентность на основе поля направлений. Когерентность является вещественным числом от 0 до 1, которое определяет степень согласованности направлений некоторой области W по формуле

$$Coh = \frac{\left|\sum W[G_{sx}\ G_{sy}]^T\right|}{\sum W\left|[G_{sx}\ G_{sy}]^T\right|} \quad (4.18)$$

Высокая когерентность означает высокую сонаправленность градиентов данной области. На основе вычисленных значений по строится поле когерентностей, которое позволяет выделить на изображении фрагменты, относящиеся к линиям отпечатков. Когерентность считается высокой, если она превосходит значение 0,5.

Как видно из формулы она представляет собой отношение модуля суммы векторов, к сумме модулей в одной окрестности. Понятно, что такая сумма не превысит единицы. И виден физический смысл такой величины: чем более сонаправлены векторы в области, тем ближе модуль их сумме к сумме модулей. И тем ближе когерентность к единице.

Знаменатель формулы считается как сумма модулей векторов градиента, в некоторой области, как показано в разделе 4.1.2.

Числитель в данной работе рассчитан, как показано в разделе 4.1.3 – модуль вектора градиента в определённой области.

Результат расчета по данной формуле представлен на рисунке 4.19.

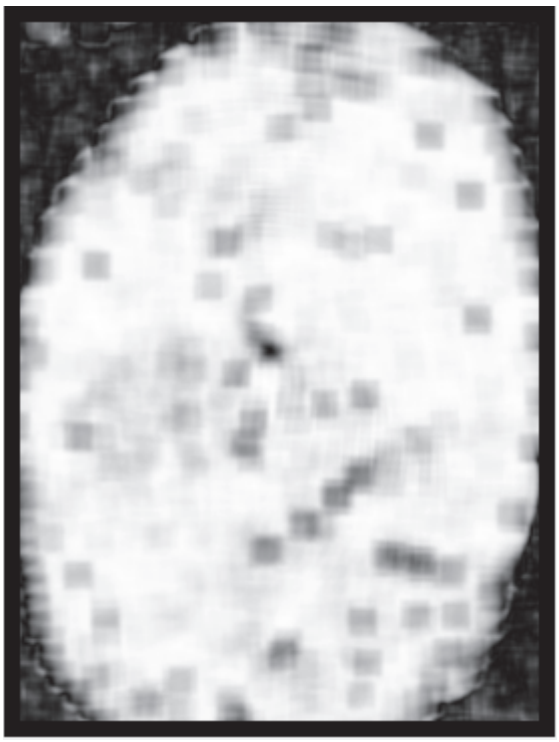


Рисунок 4.19 – Рассчитанная когерентность изображения

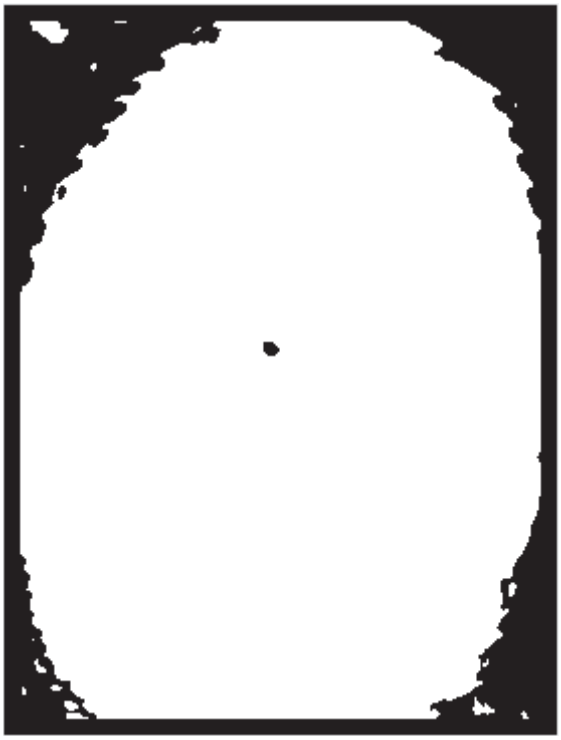


Рисунок 4.20 – Информативная область, построенная по когерентности

По рисунку 4.20 видно, что одного расчета когерентности может быть недостаточно для качественного выделения информативной области, также стоит принять в расчет, что на данный момент поле плотности и поле когерентности строились на относительно качественном и однородном отпечатке.

Но даже в таком случае остается множество краевых эффектов, способных повлиять на качество результата.

Островок, внутри области, построенный по результатам поля когерентности объясняется наличием петли на исходном изображении.

4.3.1.2 Волосы вероники

Другой показатель качества можно определить, как минимальную из амплитуд, по которым рассчитывалась плотность в окрестности в разделе 4.2.

Если внутри буфера B' нашлась только одна амплитуда, то считаем плотность линий в точке равной нулю, что даст лучшую работу алгоритма на границах отпечатка.

Рисунок, на котором отражены минимальные амплитуды для каждой окрестности называют еще созвездием «волосы Вероники». Именно это изображение также является одним из пригодных для определения информативной области. Результат представлен на рисунке 4.21.

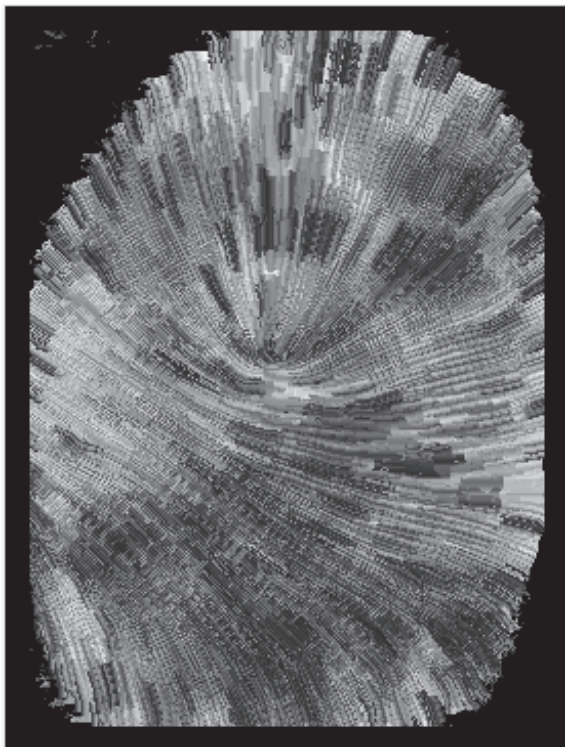


Рисунок 4.21 – Минимальная амплитуда – созвездие «волосы Вероники»

Кроме того, для улучшения качества построения информативной области можно произвести дополнительное сглаживание данного изображения. Результат сглаживания показан на рисунке 4.22.

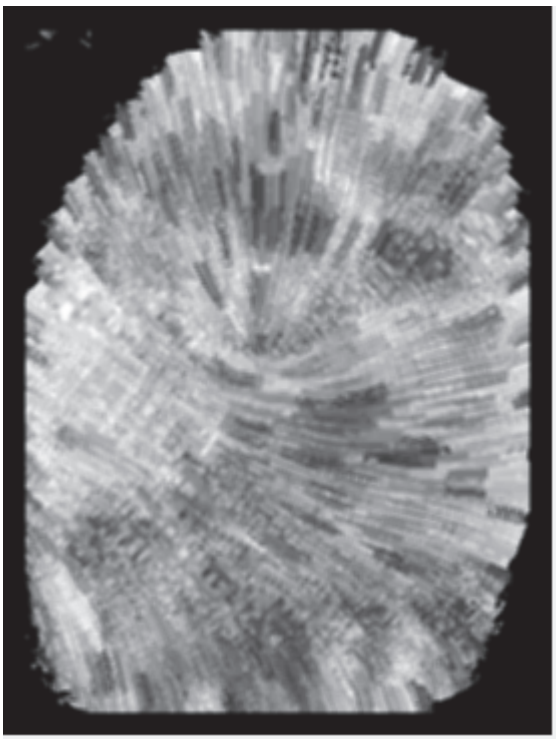


Рисунок 4.22 – Размытие изображения минимальной амплитуды

Рисунок 4.23 – Информативная область по амплитудам

Как видно по рисунку 4.23, область, построенная по изображению 4.22 также обладает некоторыми изъянами. Следовательно, на данный момент можно

сделать вывод о том, что каждый из показателей, посчитанный сам по себе дает результат с определенными помехами.

4.3.1.3 Комплексный показатель

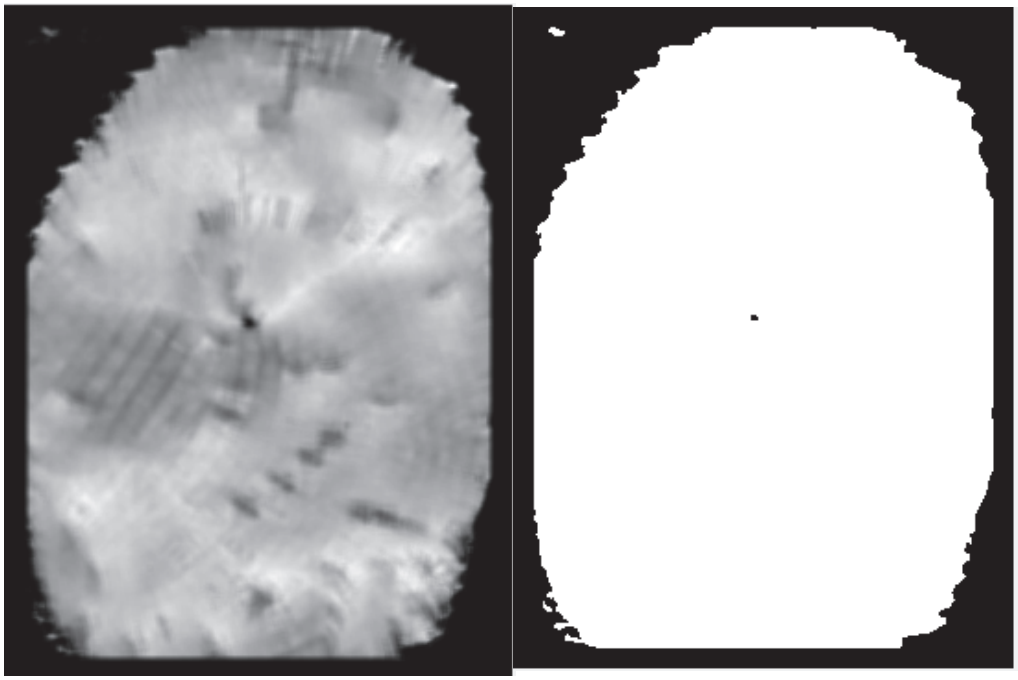
Исходя из выводов, описанных в предыдущем пункте, сделана попытка улучшения алгоритма построения информативной области путем введения комплексного показателя качества, основанного на комбинации поля плотности, и когерентности.

Идея заключается в том, что результирующий показатель будет посчитан по следующей формуле:

$$R(i) = P(i) * Sin(C(i) * 90^o) \quad (4.19)$$

Как видно из формулы, результирующий показатель является умножением поля плотности изображения на множитель с когерентностью, который тем ближе к единице, чем выше когерентность.

Результат и информативная область, построенные по формуле 4.19, показаны на рисунке 4.24.



а) б)

Рисунок 4.24 – Комплексный показатель (а), его бинаризация (б)

Как видно по рисунку 4.24, комплексный показатель дает границы информативной области наиболее близкие к их истинному положению.

4.3.1.4 Эрозия и Дилатация

Построенная информативная область может иметь рваные края или небольшие «островки», как показано на рисунках 4.20, 4.23, 4.24б.

Для устранения данных недостатков прибегнем к методам эрозии и дилатации. [24]

Для выполнения эрозии внутрь информативной области описан следующий алгоритм: на каждый пиксель изображения накладывается маска размерностью 3x3, если под ней находится только один белый пиксель, то все пиксели под маской закрашиваются черным.

Для выполнения дилатации: на каждый пиксель изображения накладывается маска размерностью 3x3, если под ней находится только один черный пиксель, то все пиксели под маской закрашиваются белым.

Эрозия и дилатация могут быть выполнены несколько раз для достижения необходимого результата. Результат применения метода представлен на рисунке 4.25.

Рисунок 4.25 – результат применения описываемых методов к рисунку 4.24(б)

4.3.1.5 Выводы по информативной области

В данном разделе было продемонстрировано, что информативная область, построенная по полю плотности обладает некоторыми недостатками на краях изображения. Рассмотрены варианты построения областей по показателям качества поля плотности, таким как когерентность, «Волосы вероники» и комплексному показателю, учитывающему, все вышеописанные и поле плотности.

В целом можно сделать вывод, что информативная область, построенная по совокупности показателей превосходит области построенные, на одном показателе. Кроме того, информативную область, построенную любым из описанных способов можно улучшить применением методов эрозии и дилатации.

4.4 Фильтрация

В данной работе фильтрация будет основана на фильтре Габора. Для этого есть несколько причин среди которых: во-первых, сам фильтр упомянут в различных источниках и хорошо понятна его основная идея. Во-вторых, фильтр хорошо подходит для периодических изображений, таких, как отпечатки пальцев.

Двумерный фильтр Габора описывается следующей формулой:

$$h(x,y) = \text{?xp}\left(-\frac{x^2+y^2}{2\sigma^2}\right) * cos2\pi f(x * sin\theta + y * cos\theta), \quad (4.19)$$

где σ – предполагаемое отклонение стандартного распределения, f – частота, θ – ориентация фильтра, x, y – координаты точки.

Как видно из формулы, для работы фильтра необходимы данные о частоте, которые найдены в разделе 4.2 и направлении – раздел 4.1.4.

Кроме того, необходимы координаты точек, в которых следует применять сам фильтр – данные из раздела 4.3.

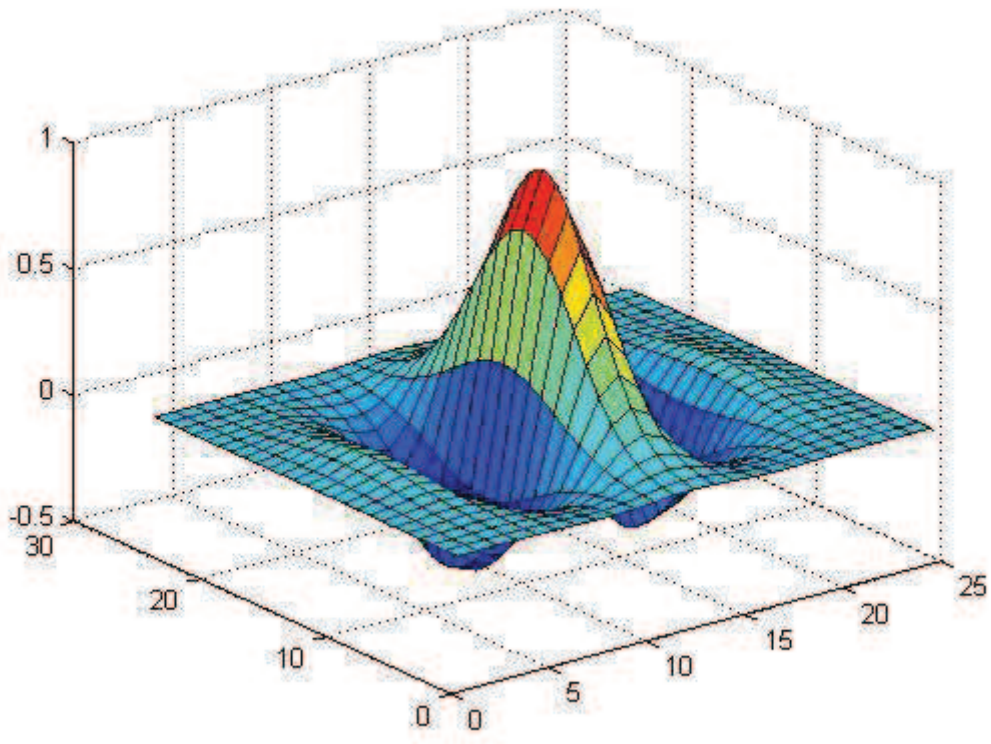


Рисунок 4.26 – График функции Габора в двумерном пространстве

Обычно фильтр Габора применяется к изображению путем наложения маски, как описано в разделе 4.1.1. Причем для его корректной работы необходимо, чтобы под маску попало несколько линий узора, поэтому размер маски следует выбирать соответствующий, порядка 15x15 пикселей. Таким образом, получается, что для обработки изображения данным фильтром напрямую для каждого пикселя необходимо посчитать 225 значений функции Габора, причем нужно учесть, что саму маску придется поворачивать и пересчитывать в зависимости от ориентации и частоты.

Существуют упрощения данного алгоритма при которых высчитываются коэффициенты маски, например, для 8 направлений, однако плотность линий всё равно меняется и необходимо производить много расчетов. Конечно можно и саму плотность линий обозначить как константную величину, но как правило, она может значительно меняться как от отпечатка к отпечатку, так и в пределах одного отпечатка.

При всем этом, после работы фильтра еще необходимо произвести бинаризацию для построения скелета изображения.

Приняв во внимание вышесказанное, можно сделать определенную модификацию алгоритма фильтрации Габора.

В целом, можно заметить, что сам фильтр имеет две составляющие. Экспоненциальная и периодическая. Первая отражает собой нормальное распределение, и если отбросить вторую составляющую, то сам фильтр выродится в фильтр гауссова размытия. Данные составляющие в одномерном случае можно видеть на рисунках 4.27 – 4.29.

Глядя на рисунок 4.26 можно заметить, что если ориентировать фильтр вдоль линий отпечатка, то в этом направлении он будет сглаживать за счет экспоненциальной составляющей, а в перпендикулярном дифференцировать и тоже сглаживать, так как поверхность фильтра таком направлении описывается одномерной функцией Габора:

$$h(x) = \exp\left(-\frac{x}{2\sigma^2}\right) * \cos(2\pi\theta x) \quad (4.20)$$

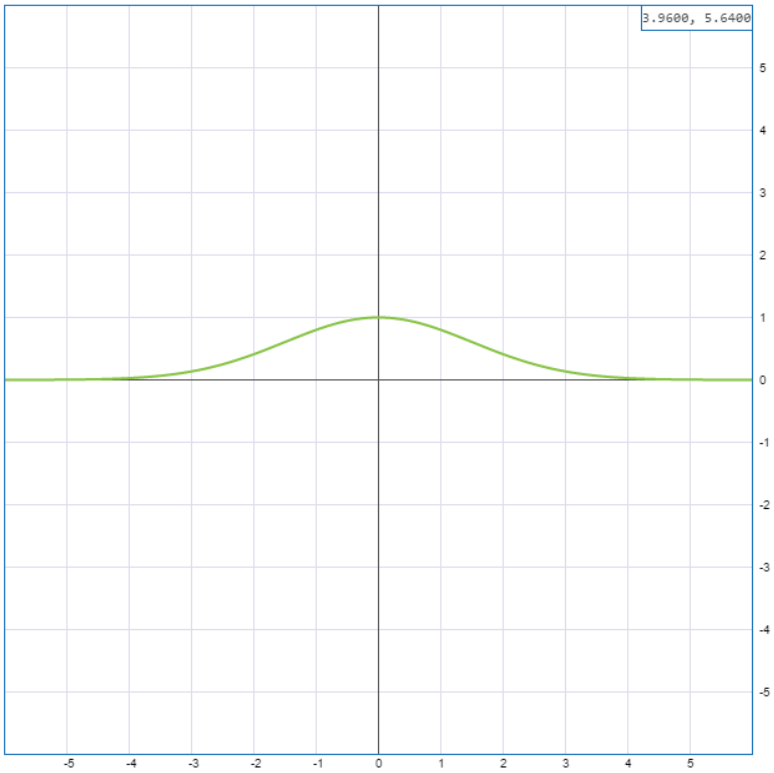


Рисунок 4.27 – График экспоненциальной составляющей функции Габора

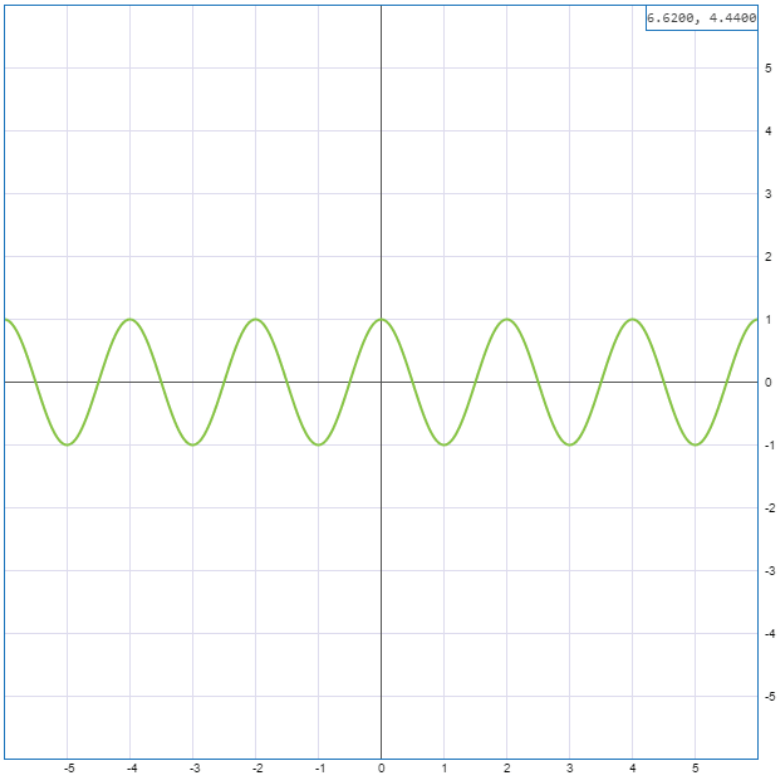


Рисунок 4.28 – График периодической составляющей функции Габора

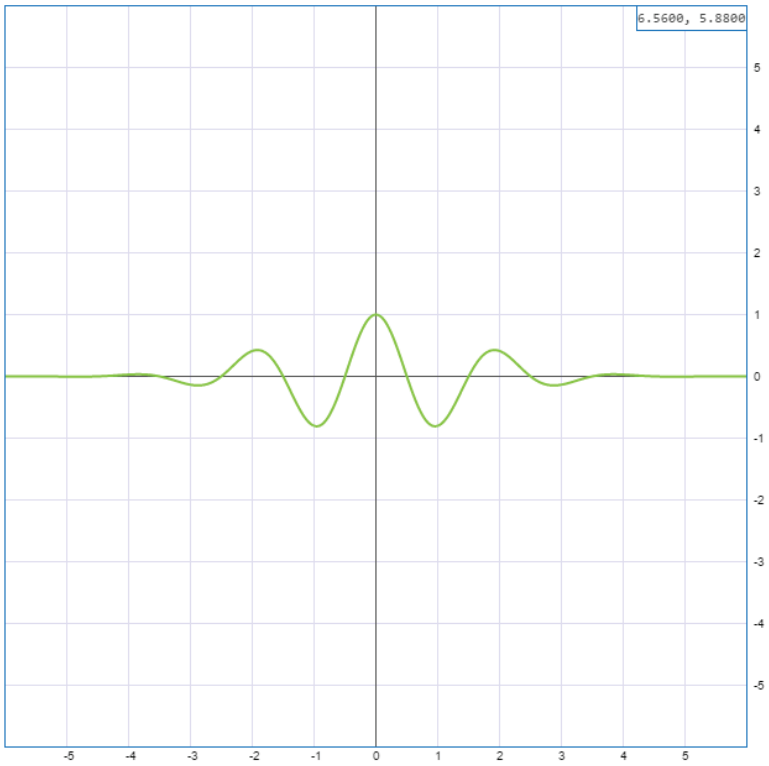


Рисунок 4.29 – График функции Габора в одномерном пространстве

На основе вышеописанных наблюдений в данной работе будет сделано следующее. Сама фильтрация разобьется на два этапа – сглаживание вдоль направления линий и дифференцирование поперек, при этом по поверхностям, совпадающими с функцией Габора.

4.4.1 Сглаживание

Воспользовавшись данными о направлении линий, полученных в разделе 4.1.4. Для каждой точки исходного изображения проведем сглаживание вдоль направления линий узора, при этом будем сглаживать не накладывая маску 15x15, а вдоль линии толщиной один пиксель, что значительно снизит вычислительные расходы.

Алгоритм сглаживания заключается в следующем – становясь в каждую точку исходного изображения, проводим линию по алгоритму Брезенхема, как было описано в разделе 4.2, с центром в обрабатываемой точке, и последовательно считываем значения яркостей пикселей, лежащих под данной линией в буфер B, размером L – рисунок 4.30.

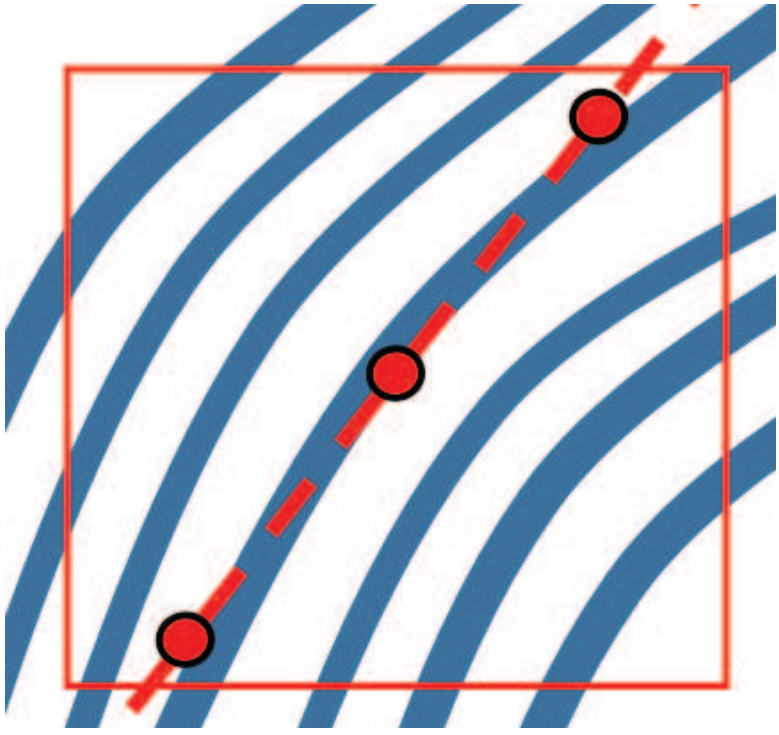


Рисунок 4.30 – Считывание вдоль направления линий

После этого каждой яркость нужно присвоить коэффициент для последующей свертки. Здесь есть два подхода: равномерный и по нормальному распределению. В первом случае каждому пикселю присваивается коэффициент 1/L. Во втором коэффициенты подбираются по графику функции равномерного распределения.

На рисунке 4.31 показан результат сглаживания вдоль направления линий

Рисунок 4.31 – Сглаживание вдоль направления линий

Как видно по рисунку 4.31 сглаживание вдоль линий также устраняет небольшие разрывы в линиях папиллярного узора, что является хорошим показателем, так как такие разрывы могут не являться постоянными признаками, и, следовательно, не должны отмечаться, как особые точки не скелете.

4.4.2 Дифференцирование

Идея дифференцирования поперек линий узора заключается в следующем: также как в предыдущем пункте, для каждой точки изображения, но уже

сглаженного, следует провести линию по алгоритму Брезенхема, и считать значения яркостей под ней в буфер B.

После чего, для каждой позиции из буфера необходимо подобрать соответствующий коэффициент.

Если считать, что размер буфера нечетный и обозначить его L, тогда его центральный элемент можно обозначить индексом 0, влево от центра значения будут последовательно убывать в целых числах, вправо – последовательно возрастать. Если сам индекс обозначить как x, тогда значения коэффициентов можно подобрать по формуле:

$$h(x) = ?\text{xp}\left(-\frac{x}{2\sigma^2}\right) * \cos(2\pi\theta x) \quad (4.21)$$

где θ – частота линий в окрестности исследуемой точки, а σ – экспериментально подобранный коэффициент. Направление можно не учитывать, так как дифференцирование уже производится поперек линий узора. Сами коэффициенты обозначим как k_x. А значения яркостей в буфере B – I_x.

После проделанной процедуры свертки на выходе будет изображение в оттенках серого, чтобы не выполнять поиск порога для бинаризации и саму процедуру бинаризации можно выполнить следующее – добиться условия

$$\sum_{x=-((l-1)/2)}^{(l-1)/2} k_x = 0 \quad (4.22)$$

Для этого после подсчета коэффициентов свертки, определить их среднее значение и вычесть из каждого коэффициента. Нормированные коэффициенты обозначим как K_x.

В таком случае после произведения свертки можно сразу строить бинаризированное изображение, при условии, что:

$$\sum_{x=-((l-1)/2)}^{(l-1)/2} K_x * I_x \leq 0 \quad (4.23)$$

покрасить пиксель черным, иначе – белым.

Результат представлен на рисунке 4.32.

Рисунок 4.32 – Результат дифференцирования

4.4.3 Выводы по фильтрации

В ходе данного раздела было сделано следующее: оригинальный фильтр Габора, разделен на две части – сглаживание вдоль локальной ориентации узора, дифференцирование поперек, по поверхностям, совпадающим с функцией Габора. Данные действия дали большой выигрыш в скорости. Для сравнения при выбранных параметрах у оригинального фильтра пришлось бы рассчитывать 225 значений функции Габора для каждой точки, а при сделанной модификации только 15, и 15 коэффициентов отдельно, которые можно посчитать перед фильтрацией.

При этом еще и автоматически выполняется бинаризация с незначительными арифметическими расчетами без использования сложных функций.

Исходный код основных методов, использовавшихся в ходе фильтрации представлен в приложении Б.

4.5 Скелетизация и выделение особых точек

4.5.1 Скелетизация

Основным предназначением любого алгоритма скелетизации является максимальное утончение всех линий на бинарном изображении, что приводит к получению скелета, в котором все линии имеют толщину в один пиксель[24], а также являются равноудаленными от границы фигуры.

После применения фильтрации и построения бинаризированного изображения на основе полученного можно построить скелет изображения.

Для построения скелета в данной работе за основу возьмем метод Зонга-Суня (Zhang-Suen), описанный Гонсалесом, где форма плоской области строится путем сведения ее к графу. Такое сокращенное представление можно получить, выделяя остов этой области с помощью алгоритма утончения. Данный метод занимает центральное место в широком классе прикладных задач обработки изображений, от автоматического контроля печатных плат до подсчета волокон асбеста в воздушных фильтрах. [11, 927с].

Учитывая, что полученное в пункте 4.4 изображение бинаризированно, это значит, что точки области имеют значение 1, а точки фона – 0. Метод состоит в последовательном применении двух шагов, применяемых для точек контура данной области. [11, 928с].

Контурной точкой будем называть любой единичный пиксель, среди восьмерки соседей которого есть хотя бы один элемент фона. Используем матрицу размером 3x3, представленную формулой 4.24.

$$\begin{vmatrix} p9 & p2 & p3 \\ p8 & p1 & p4 \\ p7 & p6 & p5 \end{vmatrix} \quad (4.24)$$

Центральная точка помечается на удаление, если выполнены следующие условия:

$$\begin{cases} 2 \le N(p_1) \le 6 \\ T(p_1) = 1 \\ p_2 * p_4 * p_6 = 0 \\ p_4 * p_6 * p_8 = 0 \end{cases} \quad (4.25)$$

N(p1) – число ненулевых соседей элемента p1, то есть число информативных – черных пикселей.

T(p1) – число переходов 0 – 1 в упорядоченной последовательности p_2, $p_3 \ldots p_8$, p_9, p_2. Это означает переход от неинформативного пикселя – белого к информативному – черному.

Два последних условия означают, что должен присутствовать хотя бы один белый пиксель из представленных.

На втором шаге первые два условия остаются такими же, а два последних заменяются на:

$$\begin{cases} 2 \le N(p_1) \le 6 \\ T(p_1) = 1 \\ p_2 * p_4 * p_8 = 0 \\ p_2 * p_6 * p_8 = 0 \end{cases} \quad (4.26)$$

Сначала к каждой точке применяется шаг 1, соответствующий формулам 4.25, если нарушается хотя бы одно из условий, то значение соответствующего элемента не меняется.

При выполнении всех условий элемент помечается на удаление, то есть должен быть закрашен в белый цвет, но сразу не удаляется, так как сначала нужно обработать все точки границы. Что позволяет предотвратить изменение структуры данных. После прохода шага 1 для всех точек границы, точки, помеченные на удаление, удаляются. Далее выполняем шаг 2 алгоритма в соответствии с формулами 4.26.

Рассмотрим случаи нарушения условий.

Первое условие на шаге 1 нарушается, если среди восьмерки соседей контурной точки p_1 имеется только один единичный – информативный элемент или только один нулевой – фоновый. Первое означает, что p_1является концевой

точкой остова, и не подлежит удалению. Удаление точки p1, имеющей семь единичных соседей, привело бы к эрозии внутри региона.

Второе условие шага 1 не соблюдается для элементов, лежащих на линии толщиной в 1 пиксель, что препятствует разделению отрезков остова.

Минимальный набор фоновых элементов в окрестности, при котором одновременно выполняются последние два этапа шага 1: (p4 = 0 или p8 = 0) или (p4 = 0 или p6 = 0). Аналогично, последние условия шага 2 одновременно выполняются для следующего минимального набора элементов фона в окрестности: (p2 = 0 или p8 = 0) или (p4 = 0 или p6 = 0). Это соответствует точкам верхней или левой границ, а также правым нижним углам границы.

Если после полного прохода алгоритма, включая оба шага, по изображению были удалены какие-либо пиксели, то необходимо повторить проход. Такое повторение должно быть до тех пор, пока во время прохода не будет удален ни один пиксель. [11, 929с].

Сложность данного алгоритма будет считаться, как O(m*n), где m, n – ширина и высота изображения, так как он совершает поточечный обход изображения в несколько итераций. В ходе алгоритма анализируются только информативные пиксели, число которых на каждой итерации уменьшается, поэтому он обладает достаточной высокой скоростью работы и быстрой сходимостью – количество итераций напрямую зависит от толщины линий фигуры.

Таким образом получаем скелет обрабатываемого изображения, но в четырехсвязной области – рисунок 4.34(а), что и является недостатком, так как при сгибах мы имеем линию толщиной не в один пиксель. Поэтому, для решения этой проблемы, далее следует перейти в восьмисвязную область, рисунок 4.34(б). Для этого используем шаблонный метод.

Для этого необходимо опытным путем подобрать шаблоны, по средствам которых будет решаться, удалять ли – закрашивать в белый центральный пиксель. Для реализации введем матрицу размерностью 3x3, элементам которой присвоим

веса, что продемонстрировано формулой 4.27. Краевые элементы имеют степени двойки от нулевой до седьмой. Каждый такой шаблон кодируется целым числом.

$$h = \begin{bmatrix} 2^7 & 2^0 & 2^1 \\ 2^6 & 0 & 2^2 \\ 2^5 & 2^4 & 2^3 \end{bmatrix} = \begin{bmatrix} 128 & 1 & 2 \\ 64 & 0 & 4 \\ 32 & 16 & 8 \end{bmatrix} \quad (4.27)$$

Если учесть, что центральный пиксель информативный, то есть черный, то существует 256 вариантов шаблона, каждый из которых индивидуален. Рисунком 4.16 продемонстрируем пример расчета. Учитывая, что изображение представлено в бинарном виде, то информативные пиксели – черные, это 1, а фон – 0. Теперь складываются значения коэффициентов матрицы тех пикселей, которые являются информативными.

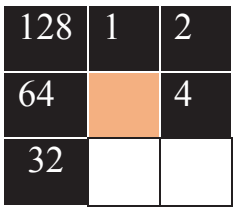
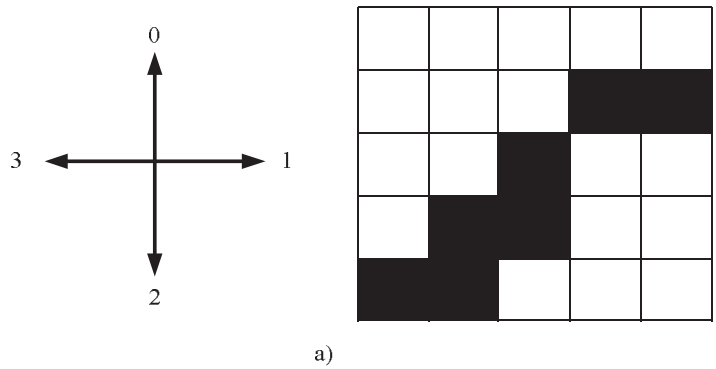
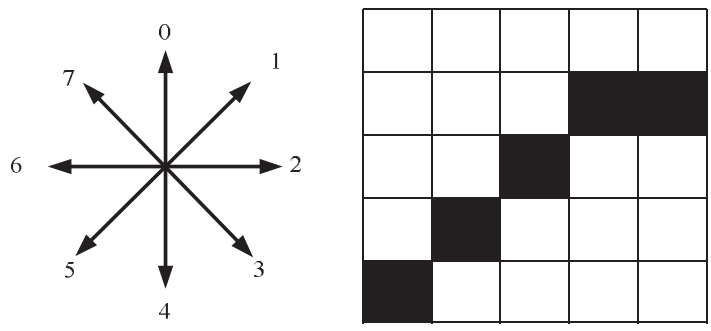


Рисунок 4.33 – Пример расчета



а)



б)

Рисунок 4.34 – Четырехсвязная область(а), восьмисвязная область(б) с примерами

В представленном примере на рисунке 4.33 расчеты производятся следующим образом:

k = 1+2+4+32+64+128 = 231. Если данный шаблон участвует в списке на удаление, то центральный пиксель окрашивается в белый.

Далее приведем несколько основных используемых шаблонов для перехода в восьмисвязную область.

Теперь представим основные виды разработанных шаблонов рисунком 4.17. Каждый из которых поворачивается до 360 градусов и заново рассчитывается.

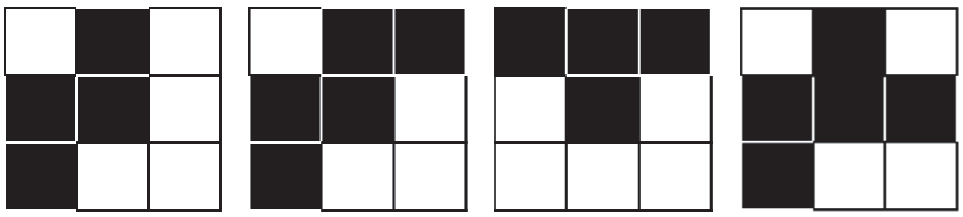


Рисунок 4.35 – Основные виды разработанных шаблонов

Таким образом мы перешли в восьмисвязную область.

Однако без дополнительной обработки, скелет получается зашумленным, много отдельных точек, небольших линий. И здесь требуются добавочные усилия, чтобы привести его к виду, пригодному для дальнейшей обработки, а именно нахождению особых точек.

Для избавления от отдельных штрихов и небольших отростков от основного скелета проведем следующую процедуру: для каждой точки, являющейся окончанием, сделаем определенное число шагов, например, около пяти. В течение каждого такого шага мы будем двигаться от точки окончания вдоль линии скелета и, если в ходе шага встретится точка ветвления, то удаляем линию до точки ветвления, а если другая точка окончания, то следует удалить всю линию. Таким образом все штрихи и отростки короче пяти точек будут удалены.

Теперь изображение пригодно для дальнейшего выделения на нем особых точек.

На рисунке 4.36 продемонстрирован конечный результат выделения остова.

Рисунок 4.36 – Скелет изображения

Стоит отметить, что благодаря применению фильтра Габора на предшествующем этапе, скелет изображения значительно улучшился, что демонстрирует рисунок 4.37.

а) б)

Рисунок 4.37 – Скелет изображения, построенный без применения фильтра Габора(а), с применением фильтра Габора(а)

4.5.2 Выделение особых точек

Для начала опишем и сравним основные существующие алгоритмы соотнесения изображений отпечатков пальцев:

- корреляционный метод;
- сравнение по узору;
- сравнение по особым точкам. [25]

Корреляционное сравнение заключается в том, что сначала накладывается полученный со сканера образ на каждый эталон из базы данных поочередно, после чего прямо по пикселям изображений осуществляется просчет различий между ними. То есть подсчитывается корреляция (по уровню интенсивности) между соответствующими пикселями, вычисленная для различных выравниваний изображений друг относительно друга (например, путем различных смещений и вращений). По соответствующему коэффициенту принимается решение об идентичности отпечатков.

Вследствие сложности и длительности работы данного алгоритма, особенно при решении задач идентификации (сравнение «один-ко-многим») – системы, построенные с его использованием, сейчас практически не используются.

Сравнение по узору. Здесь происходит сравнение непосредственно по особенностям строения папиллярного узора на поверхности пальцев. И заключается метод в том, что сначала полученное со сканера изображение образа разбивается на множество маленьких квадратных зон. В пределах каждой зоны бугорки образуют изогнутые дуги. Специальный модуль рассматривает линии в квадратиках поочередно и, каждую из них описывает уравнением синусоидальной волны, то есть задает начальный сдвиг фазы, длину волны и направление ее распространения. Соответственно при получении отпечатка для сравнения – он выравнивается и приводится к такому же виду, что и шаблон. Затем сравниваются параметры волновых представлений соответствующих ячеек. [25]

Преимуществом алгоритмов этого класса является то, что данные алгоритмы сравнения не требуют получения изображения высокого качества.

Сравнение по особым точкам строится с применением, так называемой картой с особыми точками (конечные точки и точки ветвления). Далее сравнивается эталонная карта с временной при получении образа.

В силу простоты реализации и скорости работы – алгоритмы этого класса являются наиболее распространенными.

Отдельно, для наглядности в таблице 4.1 выделены преимущества и недостатки вышеописанных методов.

Таблица 4.1. Сравнение основных методов сопоставления изображений

Название метода	Преимущества	Недостатки
Корреляционное сравнение	Низкие требования к качеству изображения отпечатка пальца.	Процесс сравнения отпечатка его пальца с эталонами должен включать в себя множество итераций, на каждой из которых изображение, полученное со сканера, поворачивается под небольшим углом или чуть-чуть смещается. Длительность операций.
Сравнение по узору	Низкие требования к качеству получаемого изображения	Высокая сложность реализации алгоритма
Сравнение по особым точкам	Быстрота работы	Высокие требования к качеству изображения (выше 300dpi)

В данной работе будем использовать метод сравнения по особым точкам. Так как используемые входные изображения отпечатков пальцев имеют

достаточно хорошее качество, пригодное для заявленного метода. Также учитывался и тот факт, что перед непосредственным выделением особых точек изображение было обработано, улучшив его качество.

Этот способ в дактилоскопии является одним из самых распространенных методов идентификации человека. По одному или нескольким изображениям отпечатков пальцев со сканера формируется шаблон, представляющий собой двухмерную поверхность, на которой выделены конечные точки, точки ветвления и точки пересечения, демонстрирующиеся на рисунке 4.38. Эти признаки являются локальными, то есть индивидуальными у каждого человека, в отличие от глобальных, таких как, вид узора. При сравнении – на отсканированном изображении отпечатка также выделяются эти точки, карта этих точек сравнивается с шаблоном и, по количеству совпавших точек принимается решение по идентичности отпечатков.

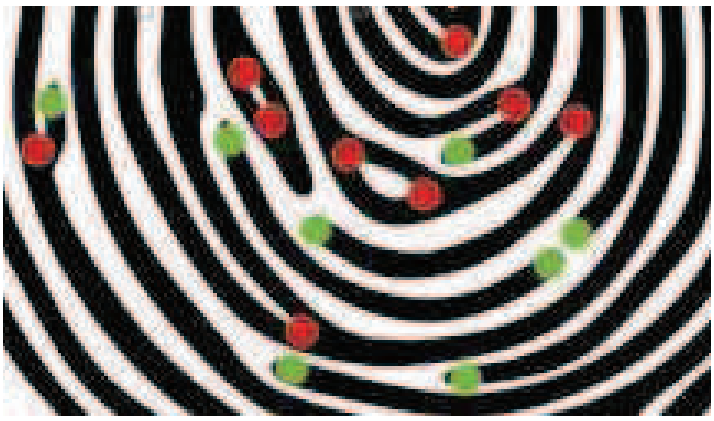


Рисунок 4.38 – Виды особых точек: окончание линии(зеленым), ветвление(красным)

В качестве особых точек изображения отпечатка пальца, различают: разветвления, пересечения и окончания. И для их нахождения воспользуемся также шаблонным методом, как и в предыдущем разделе при переходе к восьмисвязной области. Основные виды шаблонов для нахождения особых точек представлены на рисунке 4.39.они также вращаются и, каждый имеет свой вес. В данном случае осуществляется анализ окрестностей точек скелета, эти данные сравниваются с заранее заданными шаблонами. В отличие от метода, описанного ранее, здесь не происходит изменения структуры скелета, то есть ничего не

удаляется. Имеющиеся шаблоны включают в себя такой набор пикселей, которые описывают каждую из нужных особых точек.

Также процесс поиска происходит за один обход скелета изображения.

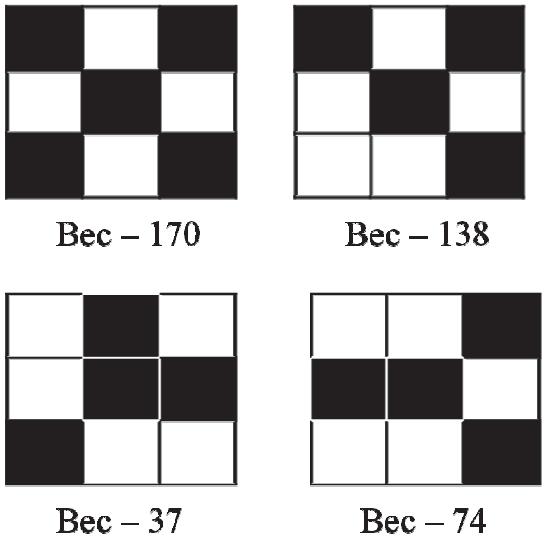


Рисунок 4.39 – Основные примеры шаблонов для нахождения особых точек

Для наглядности выделим найденные особые точки разными цветами. Результат всей проделанной работы демонстрируется рисунком 4.40.

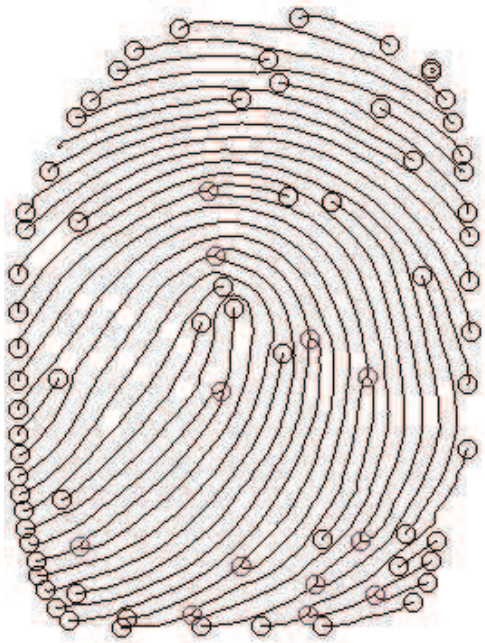


Рисунок 4.40 – Все найденные особые точки

Отметим, что окончания, которые находятся на границе информативной области, не считаются, как начало или конец исследуемой линии. Так как невозможно в обычных условиях получить два идентичных отпечатка, например, на сканере. Ведь человек каждый раз прикладывает палец по – разному, под другим углом, с разной силой надавливания, поэтому границы каждый раз различаются. Поэтому те точки, находящиеся к границе ближе, чем 10 пикселей, данный параметр подбирается вручную, считаем не особыми, следовательно, не выделяем. Учитывая этот факт, избавимся от описываемой ситуации, что демонстрируется рисунком 4.41.

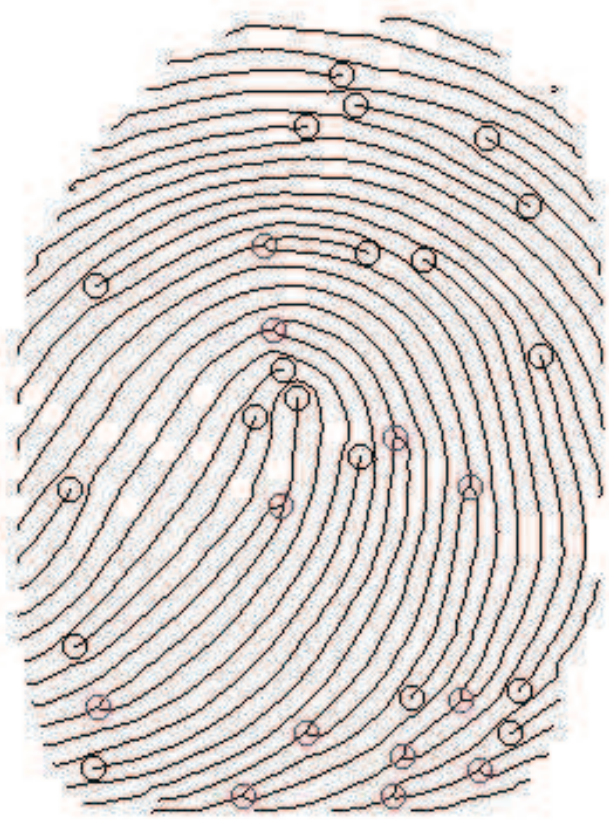


Рисунок 4.41 – Информативные особы точки

Как и говорилось ранее, два отпечатка одного пальца будут отличаться друг от друга поворотом, смещением, изменением масштаба и площадью соприкосновения в зависимости от того, как пользователь прикладывает палец к сканеру. Поэтому нельзя сказать, принадлежит ли отпечаток человеку или нет на основании простого их сравнения: векторы эталона и текущего отпечатка могут отличаться по длине, содержать несоответствующие детали и т.д.[15]

Для нахождения начала (конца) линии можно либо также использовать шаблон, примеры представленный на рисунке 4.42, либо считать количество черных окружающих пикселей по формулам 4.28 или 4.29. Если в окружении информативного пикселя находим всего один черный, то считаем исследуемый пиксель особой точкой – началом (концом).

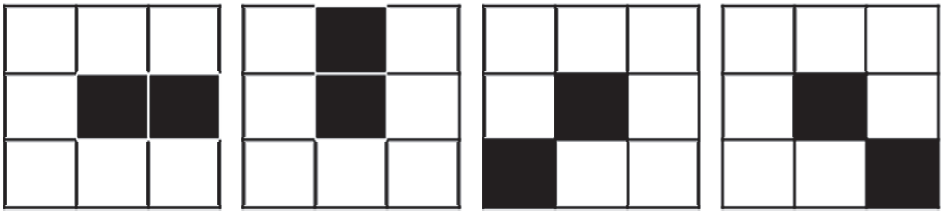


Рисунок 4.42– Основные виды шаблонов для нахождения конца линии и их веса

N1(p1) – число ненулевых соседей элемента p1, то есть число информативных – черных пикселей. Что соответствует формуле 4.28.

N2(p1) – число нулевых соседей элемента p1, то есть число неинформативных – белых пикселей. Что соответствует формуле 4.29.

Для наглядности выделим найденные особые точки разными цветами. Результат всей проделанной работы демонстрируется рисунком 4.41

Все найденные особые точки сохраняются в файл для дальнейшей работы.

Полученный на предыдущем этапе скелет изображения и на этом – особые точки являются одними из основных входных данных для последующей операции идентификации. По числам совпадений точек принимается решение процесса идентификации. Этап аутентификации по отпечатку пальца является сравнением исходных шаблонов с эталоном из базы данных по особым точкам. На основе результатов сравнений мы получим решение пропуска или отказа доступа.

Визуально проанализировав полученное изображение, можно сделать вывод, что данный метод верно выделил все особые точки, присутствующие на изображении.

4.5.3 Вывод по разделу 4.5

В данном разделе были рассмотрены основные методы скелетизации и выделения особых точек. Был усовершенствован алгоритм скелетизации Зонга-Суня для получения на изображении линии, толщиной в один пиксель. Для этого применялся шаблонный метод, с заранее разработанными шаблонами. А также были удалены короткие отростки и отдельные линии. Для последующего отыскания на полученном изображении особых точек, также применялся шаблонный метод, к которому были заранее разработаны шаблоны с разными типами нужных точек.

В результате на изображении были выделены особые точки отпечатка пальца, такие как, разветвление и окончание. Данные о координатах особых точек непосредственно будут использоваться для дальнейшего процесса верификации, где происходит сравнение шаблона, полученного нового отпечатка пальца, с имеющимися в базе.

Основные методы процедур скелетизации и выделения особых точек представлены в приложении В.

5 ПРОВЕРКА НА СООТВЕТСТВИЕ ТРЕБОВАНИЯМ ТЗ

1. Функциональные требования.

В ходе работы было реализовано приложение по обработке изображения отпечатка пальца.

Реализована фильтрация и построение бинаризированного изображения на основе исходного для последующей скелетизации. В качестве подготовительных этапов была найдена информативная область изображения, поля потоков и поля направлений.

Результаты применения методов каждого этапа обработки изображения наглядны, то есть на экран выводятся исходное изображение и обработанное. А также присутствует возможность выбора этапа в любой момент, построение гистограммы и сглаживание изображения.

Были реализованы следующие этапы обработки:

1) построение модуля градиента;
2) построение направления градиента;
3) построение модуля градиента в окрестности;
4) построение направления градиента в окрестности;
5) построение когерентности изображения, а также минимум и максимум в точке при различных размерах апертуры;
6) построение поля плотности изображения;
7) построение поля направления изображения;
8) визуализация поля направлений линиями;
9) минимум два различных показателя качества выполнения построения поля плотности;
10) визуализация информативной области;
11) дилатация, эрозия;
12) результат работы фильтра Габора (бинаризированный);
13) скелет изображения;
14) выделение особых точек.

На рисунке 5.1 представлен внешний вид разрабатываемого приложения. Также видны пункты меню.

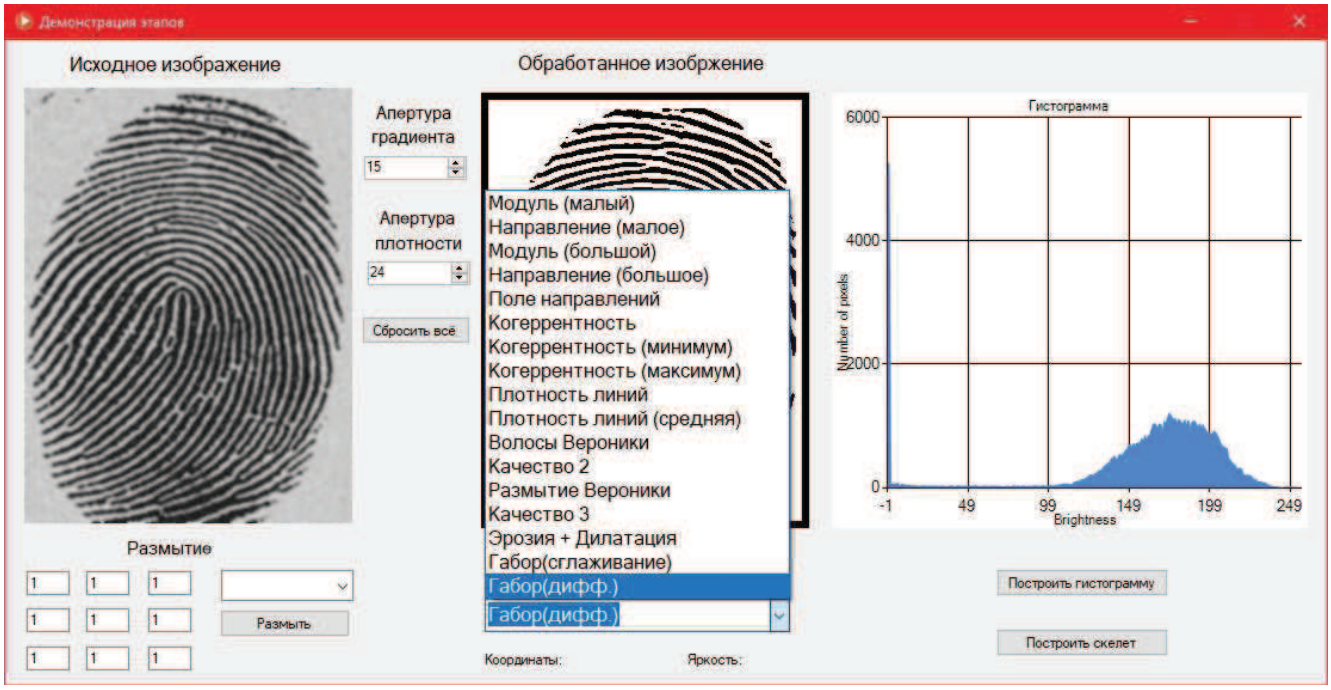


Рисунок 5.1 – Внешний вид приложения

2. Требования к ПО

Программа для демонстрации работы реализованных алгоритмов:

- написана на языке C#;
- оформлена в виде графического приложения на технологии Windows Forms.

Демонстрируемые изображения отображаются в контейнере PictureBox.

Всё время работы программы отображается исходное изображение.

Присутствует возможность сглаживания изображения произвольной матрицей 3x3 и построение гистограммы.

Для гистограммы выделена отдельная область в виде графика.

Все этапы обработки изображения отображаются рядом с исходным изображением и, а также благодаря выпадающему списку они доступны в любой момент. Уже выбранные ранее этапы не пересчитываются при повторном выборе.

Предусмотрена кнопка полного сброса. Предусмотрено введение, редактирование числовых параметров, требующихся на определенных этапах, а

также возможность задания их в программе. Кроме того, присутствует возможность получения координат положения курсора мыши на обработанном изображении.

Результаты, полученные в данном разделе, позволяют нам сделать заключение о том, что приложение полностью отвечает требованиям технического задания.

6 ЗАКЛЮЧИТЕЛЬНЫЙ РАЗДЕЛ

В ходе данной работы был разработан программный комплекс по фильтрации изображения и дальнейшего выделения особых точек на нем.

На рисунке 6.1 представлены основные проделанные этапы и их результаты.

Резюмируя выполненную работу, было сделано:

Для подсчёта поля плотности изображения были подсчитаны вектора градиента в каждой точке изображения, используя оператор Собеля. Также, были подсчитаны усредненные вектора градиентов в области, причем само усреднение было выполнено по модифицированному алгоритму, предложенного М. Базеном, в котором модули результирующих векторов вычислялись по оригинальным модулям векторов в области, а угол брался удвоенным. Что в конечном итоге дало более плавное изображение.

Используя полученные значения направлений векторов в области, а также значения модулей векторов градиента в точке, по собственному методу, основанному на анализе производной по градиенту, перпендикулярно, направлению линий узора и обработке полученных результатов по формуле, предложенной Д. Мальтони, было построено поле плотности линий дактилоскопического узора.

На основе направления векторов градиента в области было построено наглядное поле направлений.

По полю плотности была построена информативная область, кроме того, используя информацию о градиентах в области и поле плотности, были подсчитаны показатели качества, среди которых когерентность по формуле, предложенной М. Базеном, «Волосы Вероники», и собственно разработанный комплексный показатель.

По описанным показателям качества также была рассчитана информативная область.

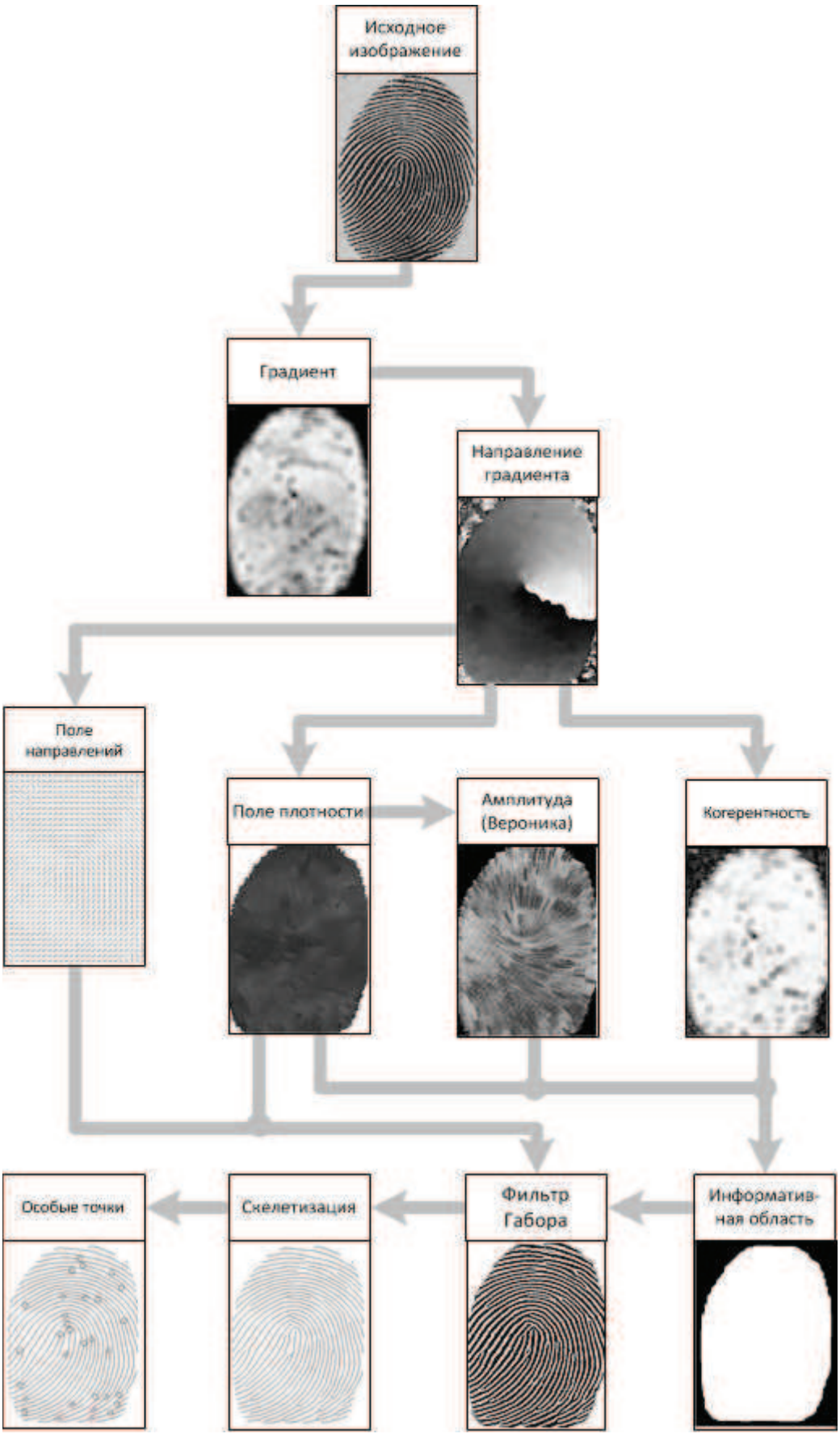


Рисунок 6.1 – Графическое представление и последовательность этапов обработки изображения отпечатка пальца

Имея информацию об информативной области, полях плотности и направлений был разработан модифицированный фильтр Габора, который разбивает на два этапа оригинальный фильтр: сглаживание и дифференцирование. Анализируя при этом не область, а только линии, сонаправленые с линиями узора в случае сглаживания, и перпендикулярные в случае дифференцирования. Причем сглаживание и дифференцирование производилось, по поверхностям, совпадающими с оригинальным фильтром Габора, что в итоге значительно уменьшило количество необходимых вычислений, но незначительно отразилось на качестве получаемого результата.

После фильтрации была выполнена процедура скелетизации изображения, основанная на алгоритме Зонга-Суня, описанным у Р. Гонсалеса. И данный метод был дополнен шаблонным методом перехода к восьмисвязной области с собственно разработанными шаблонами, что в итоге дало линии, толщиной в один пиксель. Кроме того, была произведена процедура очистки от отростков и коротких линий, на основе своего алгоритма.

На полученном скелете, также основываясь на собственных шаблонах были отмечены особые точки ветвления и окончания, при расположении таковых на определенном расстоянии от границы информативной области.

В ходе работы предложенные методы выполнялись на изображении относительно хорошего качества, для демонстрации, однако полученные результаты, хорошо применимы и для изображений значительно более низкого качества, что демонстрируется на рисунках 6.2 – 6.7.

Посмотрев на результат работы описанного алгоритма применительно к изображениям плохого качества, можно сделать вывод, что разработанный алгоритм фильтрации-скелетизации значительно улучшает качество скелета изображения по сравнению с методами без фильтрации, хорошо справляясь с входными данными низкого качества.

В целом можно сделать заключение, о том, что результаты, полученные в работе, можно применять в практических целях или для дальнейших исследований.

Рисунок 6.2 – Исходное изображение №1

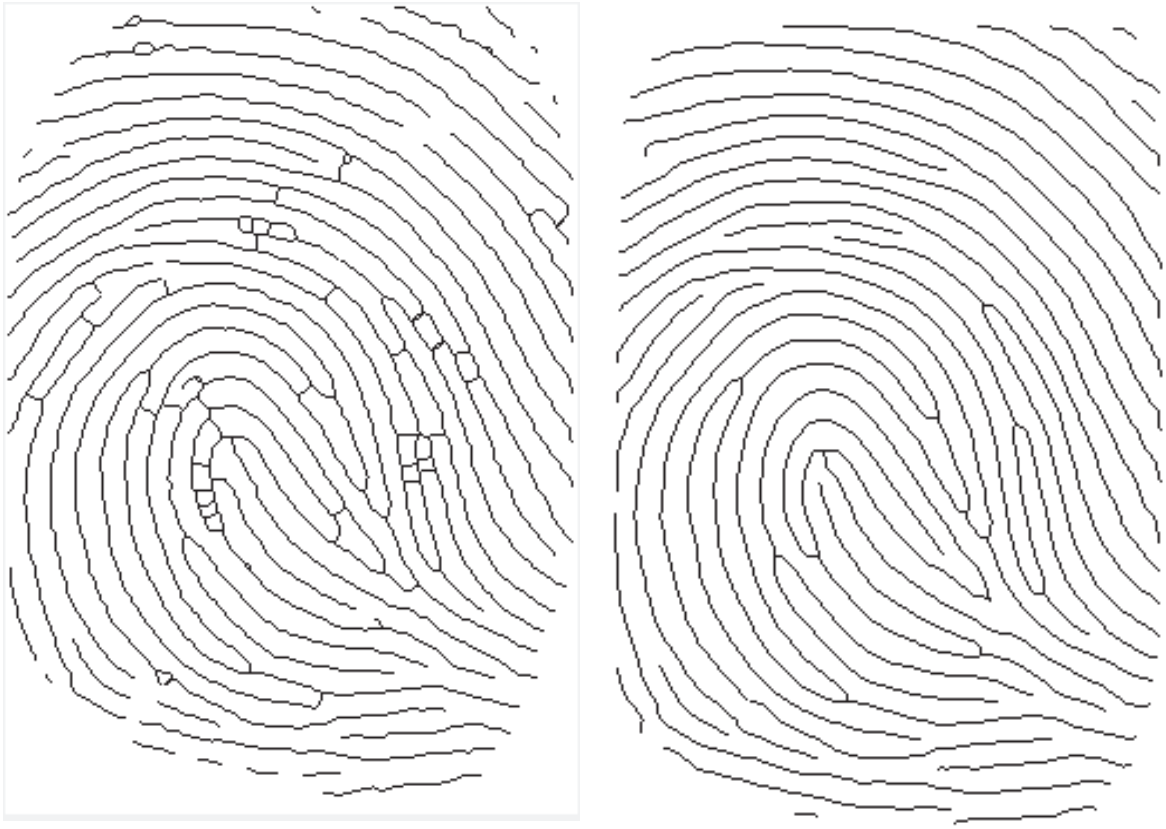


Рисунок 6.3 – Скелеты сглаженного изображения №1 без фильтрации и с ней

Рисунок 6.4 – Исходное изображение №2

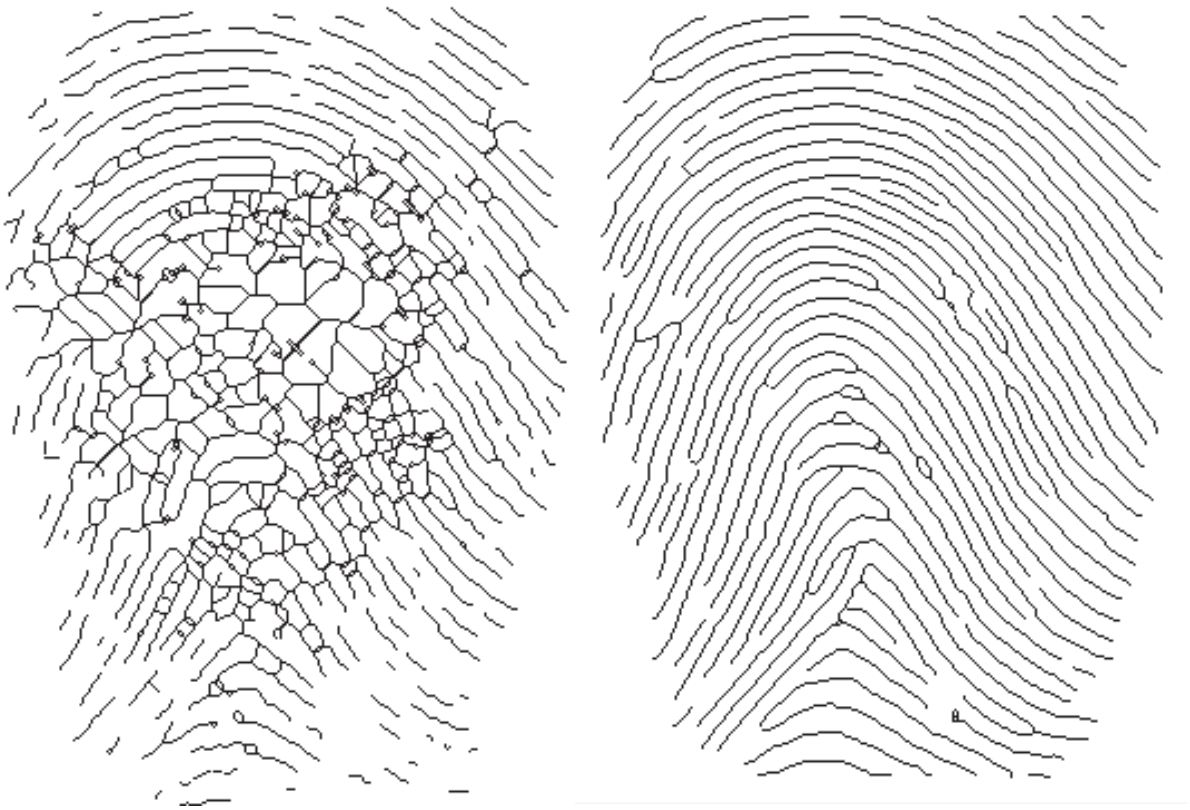


Рисунок 6.5 – Скелеты сглаженного изображения №2 без фильтрации и с ней

Рисунок 6.6 – Исходное изображение №3

Рисунок 6.7 – Скелеты сглаженного изображения №3 без фильтрации и с ней

СПИСОК ЛИТЕРАТУРЫ

1. Davide Maltoni, Dario Maio, Anil K. Jain, Salil Prabhakar. Handbook of Fingerprint Recognition. Second Edition. Springer-Verlag London Limited 2009 – 506c.
2. ЯндексМаркет. Мобильные телефоны со сканером отпечатка пальца. [Электронный ресурс]. URL: https://market.yandex.ru/. (дата обращения 27.06.2017).
3. Precise BioMatch. Технология биометрической аутентификации. [Электронный ресурс]. URL: www.morepc.ru. (дата обращения 27.06.2017).
4. BioMedSearch. Sex determination from fingerprint ridge density [Электронный ресурс]. URL: http://www.biomedsearch.com/article/Sex-determination-from-fingerprint-ridge/187844335.html (дата обращения 26.06.2017).
5. Википедия. Дактилоскопия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Дактилоскопия. (дата обращения 5.02.2017).
6. Гаспарян А.В. Система сравнения отпечатков пальцев по локальным признакам / А.В. Гаспарян, А.А. Киракосян // Вестник РАУ. Серия физико-математические и естественные науки. – 2006.
7. ScienceDirect. Determination of sex difference from fingerprint ridge density in northeastern Thai teenagers [Электронный ресурс]. URL: http://www.sciencedirect.com/science/article/pii/S2090536X15000738 (дата обращения 26.06.2017).
8. BioMedSearch. Sex determination from fingerprint ridge density [Электронный ресурс]. URL: http://www.biomedsearch.com/article/Sex-determination-from-fingerprint-ridge/187844335.html (дата обращения 26.06.2017).
9. ResearchGate. Fingerprint ridge density in the Argentinean population and its application to sex inference: A comparative study [Электронный ресурс]. URL: https://www.researchgate.net/publication/282946342_Fingerprint_ridge_density_in_the_Argentinean_population_and_its_application_to_sex_inference_A_comparative_study (дата обращения 26.06.2017).

10. Age and gender determination from finger prints using RVA and DCT coefficients [Электронный ресурс]. URL: http://www.iosrjen.org/Papers/vol3_issue8%20(part-5)/B03850509.pdf (дата обращения 26.06.2016).
11. Р. Гонсалес, Р. Вудс. Цифровая обработка изображений. – Москва: Изд-во Техносфера, 2005. – 1072 с.
12. Bazen, A.. Fingerprint Identification – Feature Extraction, Matching, and Database Search / Asker M. Bazen. – Enschede, The Netherlands: Univ. of Twente, 2002. – 187 p.
13. Задорожный В.И. Идентификация по от печаткам пальцев / В.И. Задорожный // PC Magazine/Russian Edition №2. – 2004.
14. Гудков, В.Ю. Новый метод определения градиентного поля дактилоскопических изображений / В.Ю. Гудков, А.В. Чернявский; под ред. акад. БАН А.Ф. Чернявского и д.т.н. В.В. Ревинского // Методы, алгоритмы и программное обеспечение гибких информационных технологий для автоматизированных идентификационных систем: сб. науч. ст. – Минск: БГУ, 1999. – С. 52-62.
15. Рыканов А.С.. Анализ методов распознавания отпечатков пальца. / А.С. Рыканов, В.С. Харченко // Вестник Национального аэрокосмического университета им. Н.Е. Жуковского «ХАИ». Системы обработки информации. – 2010. – №6. –7с.
16. Туркин А. В. Методы и алгоритмы обработки изображений в дактилоскопической системе на основе локальных особенностей. // Известия высших учебных заведений. Электроника. – 2012. – №1. – С. 59-66.
17. Wikipedia. Gabor filter. [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Gabor_filter. (дата обращения 27.06.2017).
18. Хабрахабр. Интересные публикации. Матричные фильтры обработки изображений. [Электронный ресурс]. URL: https://habrahabr.ru/post/142818/html (дата обращения 5.02.2017).

19. Кудрявцев Л.В. Краткий курс математического анализа – М.: Наука, 1989 – 736с
20. Хабрахабр. Интересные публикации. Алгоритмы выделения контуров изображений. [Электронный ресурс]. URL: https://habrahabr.ru/ post/114452/ (дата обращения 5.02.2017).
21. Bazen, A.. Fingerprint Identification – Feature Extraction, Matching, and Database Search / Asker M. Bazen. – Enschede, The Netherlands: Univ. of Twente, 2002. – 187 p.
22. Stackoverflow. Online community for programmers. [Электронный ресурс]. URL: http://stackoverflow.com/questions/3536428/draw-a-line-at-a-specific-angle-in-java. (дата обращения 5.02.2017).
23. Davide Maltoni. A Tutorial on Fingerprint Recognition. Biometric Systems Laboratory - DEIS - University of Bologna. URL: http://users.utcluj.ro/~tmarita/HCI/C9-extra/Fingerprints-Tutorial-Maltoni.pdf. (дата обращения 5.02.2017).
24. Яне Б. Цифровая обработка изображений. – Москва: Изд-во Техносфера, 2007. – 584 с.
25. Прогноз финансовых рисков. Идентификация по отпечаткам пальцев. [Электронный ресурс]. URL: http://bre.ru/security/21052.html. (дата обращения 27.06.2017).

ПРИЛОЖЕНИЯ

Приложение А

Целочисленный арктангенс

```
static int[] values = {
        0,  0,  0,  1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,
        3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  4,  5,  5,  5,  5,
        5,  5,  5,  6,  6,  6,  6,  6,  6,  7,  7,  7,  7,  7,  7,  7,
        8,  8,  8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9, 10, 10, 10,
        10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 12,
        12, 13, 13, 13, 13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 15,
        15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 17, 17,
        17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 18, 19, 19, 19,
        19, 19, 19, 19, 19, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21,
        21, 21, 21, 21, 21, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 23, 23,
        23, 23, 23, 23, 23, 23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
        25, 25, 25, 25, 25, 25, 25, 25, 25, 26, 26, 26, 26, 26, 26, 26,
        26, 26, 26, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 28, 28, 28,
        28, 28, 28, 28, 28, 29, 29, 29, 29, 29, 29, 29, 29, 29,
        29, 29, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 31, 31,
        31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32,
        32,
    };

 /*Integer arctangent (0-255)*/
 public static int atanInt(int y, int x){
     int abx = Math.Abs(x);
     int aby = Math.Abs(y);

     if (abx + aby == 0){
         return 0; //-1
     }

     int isNegX = (x < 0) ? 1 : 0;
     int isNegY = (y < 0) ? 1 : 0;
     int more = (aby > abx) ? 1 : 0;

     switch ((isNegY << 2) + (isNegX << 1) + (more)){
         default: return (values[at(aby, abx)]);
         case 1: return (64 - values[at(abx, aby)]);
         case 2: return (128 - values[at(aby, abx)]);
         case 3: return (64 + values[at(abx, aby)]);
         case 4: return (256 - values[at(aby, abx)]) & 255;
         case 5: return (192 + values[at(abx, aby)]);
         case 6: return (128 + values[at(aby, abx)]);
         case 7: return (192 - values[at(abx, aby)]);
     }
 }
```

Приложение Б

Основные методы фильтрации

```
/*Module and argument of gradient (at point)*/
public static void pointGrad() {
    double z1, z2, z3, z4, z5, z6, z7, z8, z9;
    double gx, gy;

    for (int x = 1; x < _width - 1; x++) {
        for (int y = 1; y < _height - 1; y++) {
            z1 = sourceImage[x - 1, y - 1];
            z2 = sourceImage[x - 1, y];
            z3 = sourceImage[x - 1, y + 1];
            z4 = sourceImage[x, y - 1];
            z5 = sourceImage[x, y];
            z6 = sourceImage[x, y + 1];
            z7 = sourceImage[x + 1, y - 1];
            z8 = sourceImage[x + 1, y];
            z9 = sourceImage[x + 1, y + 1];

            /*Sobel operator*/
            gx = (z7 + 2 * z8 + z9) - (z1 + 2 * z2 + z3);
            gy = (z3 + 2 * z6 + z9) - (z1 + 2 * z4 + z7);

            pointModule[x, y] = Math.Sqrt(gx * gx + gy * gy);
            pointAngle[x, y] = Trigon.atanDouble(gy, gx);
        }
    }
}

/*Module and argument of gradient (at area)*/
public static void areaGrad(int size) {
    int outer = (size + 1) / 2;
    int inner = (size - 1) / 2;

    for (int x = outer; x < _width - outer; x++) {
        for (int y = outer; y < _height - outer; y++) {
            double sumX = 0;
            double sumY = 0;
            double sumModules = 0;

            for (int w = x - inner; w < x + inner; w++) {
                for (int z = y - inner; z < y + inner; z++) {
                    sumX += pointModule[w, z] * Trigon.cos(2 * pointAngle[w, z]);
                    sumY += pointModule[w, z] * Trigon.sin(2 * pointAngle[w, z]);

                    sumModules += pointModule[w, z];
                }
            }
            areaModule[x, y] = Math.Sqrt(sumX * sumX + sumY * sumY);
            areaAngle[x, y] = Trigon.atanDouble(sumY, sumX) / 2;
            //Coherence
            if (sumModules == 0) {
                areaCoh[x, y] = 0;
            } else {
                areaCoh[x, y] = areaModule[x, y] / sumModules;
            }
        }
    }
}
```

```
public static double cohArea(int size, int x, int y) {
    double sumX = 0;
    double sumY = 0;
    double sumModules = 0;

    int area = (size - 1) / 2;

    for (int w = x - area; w < x + area; w++) {
        for (int z = y - area; z < y + area; z++) {
            sumX += pointModule[w, z] * Trigon.cos(2 * pointAngle[w, z]);
            sumY += pointModule[w, z] * Trigon.sin(2 * pointAngle[w, z]);

            sumModules += pointModule[w, z];
        }
    }
    return (Math.Sqrt(sumX * sumX + sumY * sumY) / sumModules);
}


public static Bitmap blur(Bitmap image, int[] filter)
{
    Bitmap resultImage = new Bitmap(image);

    double equalizer = 0;

    for (int i = 0; i < filter.Length; i++)
    {
        equalizer += filter[i];
    }
    equalizer = 1 / equalizer;

    for (int x = 1; x < _width - 1; x++)
    {
        for (int y = 1; y < _height - 1; y++)
        {
            double color = 0;

            color += sourceImage[x - 1, y - 1] * filter[0] * equalizer;
            color += sourceImage[x - 1, y] * filter[1] * equalizer;
            color += sourceImage[x - 1, y + 1] * filter[2] * equalizer;
            color += sourceImage[x, y - 1] * filter[3] * equalizer;
            color += sourceImage[x, y] * filter[4] * equalizer;
            color += sourceImage[x, y + 1] * filter[5] * equalizer;
            color += sourceImage[x + 1, y - 1] * filter[6] * equalizer;
            color += sourceImage[x + 1, y] * filter[7] * equalizer;
            color += sourceImage[x + 1, y + 1] * filter[8] * equalizer;

            Color averageColor = Color.FromArgb(Convert.ToInt32(color),
              Convert.ToInt32(color), Convert.ToInt32(color));
            resultImage.SetPixel(x, y, averageColor);
        }
    }

    return resultImage;
}
```

```
private static List<Coord> getLineCoordinates(int x, int y, double angle, int
  lineLength) {
    List<Coord> lineCoordinates;

    if (angle <= 180.0) {
        lineCoordinates = Picture.getLine(
         Convert.ToInt32(x + Trigon.cos(angle) * lineLength / 2),
         Convert.ToInt32(y + Trigon.sin(angle) * lineLength / 2),
         Convert.ToInt32(x + Trigon.cos(angle + 180) * lineLength / 2),
         Convert.ToInt32(y + Trigon.sin(angle + 180) * lineLength / 2));
    }
    else {
        lineCoordinates = Picture.getLine(
         Convert.ToInt32(x + Trigon.cos(angle) * lineLength / 2),
         Convert.ToInt32(y + Trigon.sin(angle) * lineLength / 2),
         Convert.ToInt32(x + Trigon.cos(angle - 180) * lineLength / 2),
         Convert.ToInt32(y + Trigon.sin(angle - 180) * lineLength / 2));
    }
    return lineCoordinates;
}


public static void ridgeDensity(int lineLength) {
    for (int x = lineLength / 2; x < _width - lineLength / 2; x++) {
        for (int y = lineLength / 2; y < _height - lineLength / 2; y++) {
            double angle = areaAngle[x, y];
            List<Coord> lineCoordinates = new List<Coord>();

            lineCoordinates = getLineCoordinates(x, y, angle, lineLength);

            file.WriteLine(lineCoordinates.Count);

            density[x, y] = calcDencity(lineCoordinates, x, y);
        }
    }
}


private static double calcDencity(List<Coord> lineCoordinates, int x, int y){
    List<int> maximas = new List<int>();
    List<double> derivative = new List<double>();
    List<double> amplitude = new List<double>();

    for (int i = 1; i < lineCoordinates.Count - 1; i++) {
        double left = pointModule[lineCoordinates[i - 1].getX(), lineCoordinates[i -
        1].getY()];
        double right = pointModule[lineCoordinates[i + 1].getX(), lineCoordinates[i +
       1].getY()];

        derivative.Add(right-left);
    }

    for (int i = 1; i < derivative.Count - 1; i++) {
        double left = derivative[i - 1];
        double center = derivative[i];
        double right = derivative[i +1];

        if (center > left && center > right && center > 60) {
            maximas.Add(i);
            amplitude.Add(center);
        }
    }
```

```
        int delimeter = maximas.Count - 1;
        int divider = 0;

        for (int i = 0; i < maximas.Count - 1; i++) {
            divider += Math.Abs(maximas[i] - maximas[i + 1]);
        }

        double dencity = (double)delimeter / divider; //Frequency

        if (divider == 0) { dencity = 0; }

        if (amplitude.Count > 1) {
            qualityDencity[x,y] = amplitude.Min();
        }

        return dencity;
    }

    public static void gaborV2(int lineLength)
    {
        //Blur along ridges direction
        for (int x = lineLength / 2; x < _width - lineLength / 2; x++)
        {
            for (int y = lineLength / 2; y < _height - lineLength / 2; y++)
            {

                double info = infoArea[x, y];

                if (info == 255)
                {
                    double angle = areaAngle[x, y];
                    if (angle < 270)
                    {
                        angle = angle + 90;
                    }
                    else angle = angle - 90;

                    List<Coord> lineCoordinates = new List<Coord>();

                    lineCoordinates = getLineCoordinates(x, y, angle, lineLength);

                    double sum = 0.0d;

                    for (int i = 0; i < lineCoordinates.Count; i++)
                    {
                          sum += constSourceImage[lineCoordinates[i].getX(),
                          lineCoordinates[i].getY()];
                    }

                    gaborBlur[x, y] = sum / lineCoordinates.Count;
                }
                else {
                    gaborBlur[x, y] = 210;
                }
            }
        }
    }
```

```
public static void gaborV2Plus(int lineLength) {
    int start = (lineLength-1)/2;

    //Differentiation across ridges direction
    for (int x = lineLength / 2; x < _width - lineLength / 2; x++) {
        for (int y = lineLength / 2; y < _height - lineLength / 2; y++) {
            double info = infoArea[x, y];

            if (info == 255) {
                double angle = areaAngle[x, y];

                double sum = 0.0d;
                double sigma = 2.5d;
                //double theta = 0.4d;
                double theta = averageDensity[x, y];
                double[] coeffs = new double[lineLength];

                for (int i = -start; i < start + 1; i++)
                {
                    double left = Math.Exp(-((i * i) / (2.0d * sigma * sigma)));
                    double right = Math.Cos(2.0d * Math.PI * i * theta);
                    coeffs[i + start] = left * right;
                    sum = sum + coeffs[i + start];
                }

                double average = sum / ((start * 2) + 1);

                double bigSum = 0.0d;


                List<Coord> lineCoordinates = new List<Coord>();
                lineCoordinates = getLineCoordinates(x, y, angle, lineLength);

                for (int i = 0; i < lineCoordinates.Count; i++){
                    bigSum += gaborBlur[lineCoordinates[i].getX(),
                     lineCoordinates[i].getY()] * (coeffs[i] - average);
                }

                if (bigSum > 0) {
                    bigSum = 255;
                } else {
                    bigSum = 0;
                }

                gaborDiff[x, y] = bigSum;
            }
            else
            {
                gaborDiff[x, y] = 255;
            }
        }
    }
}
```

Приложение В

Основные методы скелетизации и выделения особых точек

```
        public static List<Coordinates> firstStep(double[,] binary)
        {
            coordinates = new List<Coordinates>();
            bool z1, z2, z3, z4, z5, z6, z7, z8, z9;

            for (int x = 1; x < width - 1; x++)
            {
                for (int y = 1; y < height - 1; y++)
                {
                    z1 = toBool(binary[x, y]);
                    z2 = toBool(binary[x, y - 1]);
                    z3 = toBool(binary[x + 1, y - 1]);
                    z4 = toBool(binary[x + 1, y]);
                    z5 = toBool(binary[x + 1, y + 1]);
                    z6 = toBool(binary[x, y + 1]);
                    z7 = toBool(binary[x - 1, y + 1]);
                    z8 = toBool(binary[x - 1, y]);
                    z9 = toBool(binary[x - 1, y - 1]);

                    bool[] mas = {/*z1,*/ z2, z3, z4, z5, z6, z7, z8, z9 };
                    if (z1 == false) //For area pixels
                    {
                        if (isBorderPixel(mas))
                        { //That's also border pixels
                            if (checkFirstStepRules(mas)) //If validate is succeed add them to
list

                            {
                                coordinates.Add(new Coordinates(x, y));
                            }
                        }
                    }
                }
            }
            return coordinates;
        }

        public static List<Coordinates> secondStep(double[,] binary)
        {
            coordinates = new List<Coordinates>();

            bool z1, z2, z3, z4, z5, z6, z7, z8, z9;

            for (int x = 1; x < width - 1; x++)
            {
                for (int y = 1; y < height - 1; y++)
                {
                    z1 = toBool(binary[x, y]);
                    z2 = toBool(binary[x, y - 1]);
                    z3 = toBool(binary[x + 1, y - 1]);
                    z4 = toBool(binary[x + 1, y]);
                    z5 = toBool(binary[x + 1, y + 1]);
                    z6 = toBool(binary[x, y + 1]);
                    z7 = toBool(binary[x - 1, y + 1]);
                    z8 = toBool(binary[x - 1, y]);
                    z9 = toBool(binary[x - 1, y - 1]);

                    bool[] mas = {/*z1,*/ z2, z3, z4, z5, z6, z7, z8, z9 };
```

```
                if (z1 == false)
                {
                    if (isBorderPixel(mas))
                    {
                        if (checkSecondStepRules(mas))
                        {
                            coordinates.Add(new Coordinates(x, y));
                        }
                    }
                }
            }
        }
        return coordinates;
    }

    private static double[,] deletePixels(double[,] binary, List<Coordinates> coordinates)
    {
        foreach (Coordinates c in coordinates) {
            binary[c.getX(), c.getY()] = 255;
        }
        return binary;
    }

    private static bool checkFirstStepRules(bool[] mas) {
        if (checkFirstRule(mas) &&
            checkSecondRule(mas) &&
            checkThirdRuleV1(mas) &&
            checkFourthRuleV1(mas)) {
            return true;
        }
        else return false;
    }

    private static bool checkSecondStepRules(bool[] mas) {
        if (checkFirstRule(mas) &&
            checkSecondRule(mas) &&
            checkThirdRuleV2(mas) &&
            checkFourthRuleV2(mas)) {
            return true;
        }
        else return false;
    }

    private static bool checkFirstRule(bool[] mas)
    {
        //Проверить число ненулевых элементов (черных)
        int areaCount = 0;
        for (int i = 0; i < mas.Length; i++)
        {
            if (mas[i] == false) areaCount++;
        }
        if ((areaCount >= 2) && (areaCount <= 6))
        {
            return true;
        }
        else return false;
    }
```

```
private static bool checkSecondRule(bool[] mas) {
    //Count number of jumps background -> area
    int jumpCount = 0;
    for (int i = 0; i < mas.Length - 1; i++)
    {
        if ((mas[i] == true) && (mas[i + 1] == false))
        {
            jumpCount++;
        }
    }
    if ((mas[mas.Length - 1] == true) && (mas[0] == false))
    {
        jumpCount++;
    }
    if (jumpCount == 1)
    {
        return true;
    }
    else return false;
}
private static bool checkThirdRuleV1(bool[] mas){
    if (mas[0] || mas[2] || mas[4]) return true;
    else return false;
}
private static bool checkFourthRuleV1(bool[] mas){
    if (mas[2] || mas[4] || mas[6]) return true;
    else return false;
}
private static bool checkThirdRuleV2(bool[] mas){
    if (mas[0] || mas[2] || mas[6]) return true;
    else return false;
}
private static bool checkFourthRuleV2(bool[] mas){
    if (mas[0] || mas[4] || mas[6]) return true;
    else return false;
}

private static bool toBool(double x){
    return x != 0.0d;
}

private static bool isBorderPixel(bool[] mas)
{
    for (int i = 0; i < mas.Length; i++)
    {
        if (mas[i] == true)
        {
            return true;
        }
    }
    return false;
}

private static Bitmap lightSpecialPointsV2(Bitmap image)
{
    List<Coord> coordinatesEnd = new List<Coord>();
    List<Coord> coordinatesBranch = new List<Coord>();

    float z1, z2, z3, z4, z5, z6, z7, z8, z9;
    bool b1, b2, b3, b4, b5, b6, b7, b8, b9;
    Color red = Color.FromArgb(255, 0, 0);
    Color green = Color.FromArgb(0, 255, 0);
    Pen redPen = new Pen(Color.Red, 1);
```

```
Pen blackPen = new Pen(Color.Black, 1);

for (int x = 1; x < image.Width - 1; x++)
{
    for (int y = 1; y < image.Height - 1; y++)
    {
        z1 = image.GetPixel(x, y).GetBrightness();
        z2 = image.GetPixel(x, y - 1).GetBrightness();
        z3 = image.GetPixel(x + 1, y - 1).GetBrightness();
        z4 = image.GetPixel(x + 1, y).GetBrightness();
        z5 = image.GetPixel(x + 1, y + 1).GetBrightness();
        z6 = image.GetPixel(x, y + 1).GetBrightness();
        z7 = image.GetPixel(x - 1, y + 1).GetBrightness();
        z8 = image.GetPixel(x - 1, y).GetBrightness();
        z9 = image.GetPixel(x - 1, y - 1).GetBrightness();

        if ((z2 + z3 + z4 + z5 + z6 + z7 + z8 + z9 == 7) && (z1 == 0))
        {
            if (info[x,y] == 255)
            coordinatesEnd.Add(new Coord(x, y));
        }

        b1 = toBool(image.GetPixel(x, y).GetBrightness());
        b2 = toBool(image.GetPixel(x, y - 1).GetBrightness());
        b3 = toBool(image.GetPixel(x + 1, y - 1).GetBrightness());
        b4 = toBool(image.GetPixel(x + 1, y).GetBrightness());
        b5 = toBool(image.GetPixel(x + 1, y + 1).GetBrightness());
        b6 = toBool(image.GetPixel(x, y + 1).GetBrightness());
        b7 = toBool(image.GetPixel(x - 1, y + 1).GetBrightness());
        b8 = toBool(image.GetPixel(x - 1, y).GetBrightness());
        b9 = toBool(image.GetPixel(x - 1, y - 1).GetBrightness());

        bool[] mas = { b2, b3, b4, b5, b6, b7, b8, b9 };

        if (b1 == false)
        {
            int number = getNumber(mas);
            if ((number == 74) ||
                (number == 164) ||
                (number == 146) ||
                (number == 41) ||
                (number == 170) ||
                (number == 57) ||
                (number == 73) ||
                (number == 148) ||
                (number == 82) ||
                (number == 138) ||
                (number == 167))
            {
                coordinatesBranch.Add(new Coord(x, y));
            }

        }
    }
}

using (var graphics = Graphics.FromImage(image)) {
    foreach (Coord c in coordinatesEnd) {
        graphics.DrawEllipse(blackPen, c.getX()-5, c.getY()-5, 10, 10);
    }

    foreach (Coord c in coordinatesBranch) {
        graphics.DrawEllipse(redPen, c.getX()-5, c.getY()-5, 10, 10);
    }
```

```
        }
        return image;
    }


    private static Bitmap markSpecialPoints(Bitmap binaryImage)
    {
        float z1, z2, z3, z4, z5, z6, z7, z8, z9;
        Color white = Color.FromArgb(255, 255, 255);
        Color red = Color.FromArgb(255, 0, 0);
        Color green = Color.FromArgb(0, 255, 0);
        for (int x = 1; x < binaryImage.Width - 1; x++)
        {
            for (int y = 1; y < binaryImage.Height - 1; y++)
            {
                z1 = binaryImage.GetPixel(x, y).GetBrightness();
                z2 = binaryImage.GetPixel(x, y - 1).GetBrightness();
                z3 = binaryImage.GetPixel(x + 1, y - 1).GetBrightness();
                z4 = binaryImage.GetPixel(x + 1, y).GetBrightness();
                z5 = binaryImage.GetPixel(x + 1, y + 1).GetBrightness();
                z6 = binaryImage.GetPixel(x, y + 1).GetBrightness();
                z7 = binaryImage.GetPixel(x - 1, y + 1).GetBrightness();
                z8 = binaryImage.GetPixel(x - 1, y).GetBrightness();
                z9 = binaryImage.GetPixel(x - 1, y - 1).GetBrightness();

                if ((z2 + z3 + z4 + z5 + z6 + z7 + z8 + z9 == 7) && (z1 == 0))
                {
                    binaryImage.SetPixel(x, y, red); //Точки конца
                }

                if ((z2 + z3 + z4 + z5 + z6 + z7 + z8 + z9 == 5) && (z1 == 0))
                {
                    binaryImage.SetPixel(x, y, green); //Точки ветвления
                }
            }
        }
        return binaryImage;
    }

    private static double[,] to8(double[,] binary)
    {
        bool z1, z2, z3, z4, z5, z6, z7, z8, z9;

        for (int x = 1; x < width - 1; x++)
        {
            for (int y = 1; y < height - 1; y++)
            {
                z1 = toBool(binary[x, y]);
                z2 = toBool(binary[x, y - 1]);
                z3 = toBool(binary[x + 1, y - 1]);
                z4 = toBool(binary[x + 1, y]);
                z5 = toBool(binary[x + 1, y + 1]);
                z6 = toBool(binary[x, y + 1]);
                z7 = toBool(binary[x - 1, y + 1]);
                z8 = toBool(binary[x - 1, y]);
                z9 = toBool(binary[x - 1, y - 1]);

                bool[] mas = { z2, z3, z4, z5, z6, z7, z8, z9 };

                if (z1 == false)
                {
                    int number = getNumber(mas);
                    if ((number == 208) ||
```

```
                    (number == 22) ||
                    (number == 97) ||
                    (number == 52) ||
                    (number == 13) ||
                    (number == 88) ||
                    (number == 67) ||
                    (number == 133) ||
                    (number == 5) ||
                    (number == 20) ||
                    (number == 80) ||
                    (number == 65))
                        binary[x, y] = 255;
                }
            }
        }
        return binary;
    }

    private static Bitmap delBranches(Bitmap binaryImage)
    {
        Color red = Color.FromArgb(255, 0, 0);
        for (int x = 1; x < binaryImage.Width - 1; x++)
        {
            for (int y = 1; y < binaryImage.Height - 1; y++)
            {
                //Для каждого найденного красного пикселя (окончание линии)
                if (binaryImage.GetPixel(x, y) == red)
                {
                    findPath(x, y, binaryImage);
                }
            }
        }
        return binaryImage;
    }



    private static int getNumber(bool[] mas)
    {
        int count = 0;
        for (int i = 0; i < mas.Length; i++)
        {
            if (mas[i] == false)
            {
                count = count += pow(i);
            }
        }
        return count;
    }

    private static int pow(int var) {
        if (var == 0) return 1;
        if (var == 1) return 2;
        if (var == 2) return 4;
        if (var == 3) return 8;
        if (var == 4) return 16;
        if (var == 5) return 32;
        if (var == 6) return 64;
        if (var == 7) return 128;
        if (var == 8) return 256;
        if (var == 9) return 512;
        return -1;
    }
```