МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.04.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Математическое моделирование
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Исследование эффективности дескрипторов особых точек на различных типах изображений»

Обучающийся Д.В. Лопатин
(Инициалы Фамилия) (личная подпись)

Руководитель к. ф.-м. н. О.В. Лелонд
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2022

Содержание

Введение

Основной задачей компьютерного зрения является распознавание различных объектов на изображениях. В настоящее время много нерешенных проблем кроется в области распознавания лиц: например, алгоритмы плохо распознают лица в условиях низкой освещённости или лица с окклюзиями (очки, маски, шапки). За точность распознавания лица на изображениях отвечают специальные дескрипторы особых точек, которые находят и описывают ключевые черты конкретного лица. Различные алгоритмы с разным успехом справляются с той или иной проблемой, но единого решения до сих пор не найдено. Более того, при разработке и публикации нового дескриптора, авторы не указывают его эффективность относительно других алгоритмов, заставляя пользователей самому выяснять это опытным путем.

Актуальность и научная значимость настоящего исследования заключается в отсутствии готовой модели для быстрого сравнения многочисленных дескрипторов особых точек.

Объектом исследования является сравнительный анализ дескрипторов особых точек.

Предметом исследования является оптимизация процесса сравнения дескрипторов особых точек.

Целью исследования является разработка подхода для сравнения эффективности работы дескрипторов особых точек на различных наборах изображений.

Гипотеза исследования состоит в том, что дескрипторы особых точек можно объективно и быстро сравнить, если будет разработана и реализована соответствующая математическая модель для их исследования.

Для достижения поставленной цели необходимо решить следующие задачи:

– проанализировать состояние предметной области – алгоритмов сравнения дескрипторов особых точек;

– обозначить и сформулировать задачи для решения найденных проблем;
– проанализировать алгоритмы работы и сформулировать математические модели дескрипторов особых точек;
– составить собственную математическую модель для сравнительного анализа дескрипторов особых точек;
– проанализировать существующее программное обеспечение для решения поставленных задач;
– выбрать набор программ и библиотек для реализации алгоритма сравнения дескрипторов особых точек;
– реализовать алгоритм для исследования эффективности особых точек в виде программного кода;
– сформулировать и проанализировать полученные результаты.

Методы исследования: анализ дескрипторов особых точек, вычислительный эксперимент на основе построенной модели.

Научная новизна исследования заключается в использовании нового алгоритма сравнения дескрипторов особых точек.

Работа состоит из введения, 3 разделов, заключения, содержит 56 рисунков, 6 таблиц, 12 формул, список использованной литературы. Основной текст работы изложен на 73 страницах.

1 Анализ состояния вопроса

1.1 Обзор предметной области

В настоящее время распознавание лиц - биометрический метод, при котором сбор данных может выполняться как при участии анализируемого лица, так и без него. Лицо можно рассматривать как самый простой способ распознать человека, что увеличивает признание такого рода систем и их приложений. Эти системы состоят из двух задач: проверки личности, когда система проверяет, соответствует ли личность человека той, которой он или она утверждает, и задачи идентификации, когда система определяет личность человека среди всех людей в базе данных. Таким образом, задача распознавания лиц охватывает обе подзадачи - аутентификацию и идентификацию [11].

Более того распознавание лиц является широко распространенным биометрическим методом для правоохранительных органов [5]. Фактически, сегодня это наиболее широко используемый биометрический метод. Распознавание лиц можно использовать для идентификации людей, а также для обнаружения поддельных идентификационных данных. Его можно использовать для идентификации отдельных лиц и групп путем сравнения лиц с их водительскими правами, паспортами и т. д [7]. Технология распознавания лиц может использоваться во многих сценариях, от фотографирования человека и проверки его по биометрической базе данных до идентификации человека в толпе или в аэропорту путем сканирования лица по базе данных [1]. Эта технология широко применяется правительственными учреждениями, банковскими учреждениями, банками, авиакомпаниями, больницами и т. д. Фактически, теперь она используется во многих общественных местах и является частью повседневной жизни [10].

Несмотря на высокую развитость систем распознавания лиц, осталось много проблем, которые пытаются решить новые алгоритмы. Вот некоторые из них:

- слабое освещение,
- частичные окклюзии изображения,
- изменения в мимике.

Все эти параметры могут значительно снижать качество распознавания лиц на изображениях. Для уменьшения погрешности в условиях слабого освещения начали использоваться специальные алгоритмы гистограммного выравнивания с ограничением контраста, например, CLAHE [3].

Исследователям из Государственного университета Сан-Паулу (UNESP) удалось найти оптимальный способ уменьшить оптический шум при распознавании лиц: интерполировать данные между двумя снимками. Для этого они полагаются на математическую форму геометрического анализа, называемую деформируемой решеткой [23]. Этот метод позволяет им решить проблему видения нескольких лиц на одном изображении без оптического шума.

Существуют разные способы минимизировать оптический шум на одном изображении [8]. Исследователи рассматривают параметр, называемый чередованием, который состоит из двух снимков, в которых выравнивание гистограммы ограничено, другими словами, точки интерполяции данных расположены от экстремума между двумя выстрелами.

По словам авторов, этот новый метод был создан путем использования двух реализаций монохроматического изображения (то есть двух размытых изображений) и свойств осей x и y. Данные накладываются на экран компьютера, а затем к изображению применяется математический алгоритм для интерполяции точек. Когда точки умножаются друг на друга, получается ноль, то есть изображение, которое мы увидели бы, когда свет был выключен, а все остальное уже было на месте. Точки можно сравнить с точкой 0,5,

которая должна появиться на увеличенном изображении после первой экспозиции.

На рисунке 1 показаны примеры разных типов освещённости лица. В настоящее время алгоритмы плохо справляются с резким изменением освещения на лицах, распознавая одно и то же лицо как несколько разных лиц.

Рисунок 1 – Пример изменения освещённости лица

Ещё одной проблемой являются многочисленные окклюзии на изображениях. Если человек надевает головной убор, шарф или очки, точность распознавания значительно падает. Решить эту проблему призваны специальные окклюзионные маски. На рисунке 2 показаны примеры различных окклюзий на изображениях.

Рисунок 2 – Окклюзии на изображениях лиц

На рисунке 3 представлен алгоритм работы системы распознавания лиц. На первом этапе входные изображения масштабируются и сохраняются в базу данных. Далее происходит извлечение и описание особых точек с помощью соответствующих дескрипторов. Именно этот этап будет более детально рассматриваться в дальнейшей работе. В конце концов на основе найденных особых точек строится и обучается модель на основе нейронных сетей.

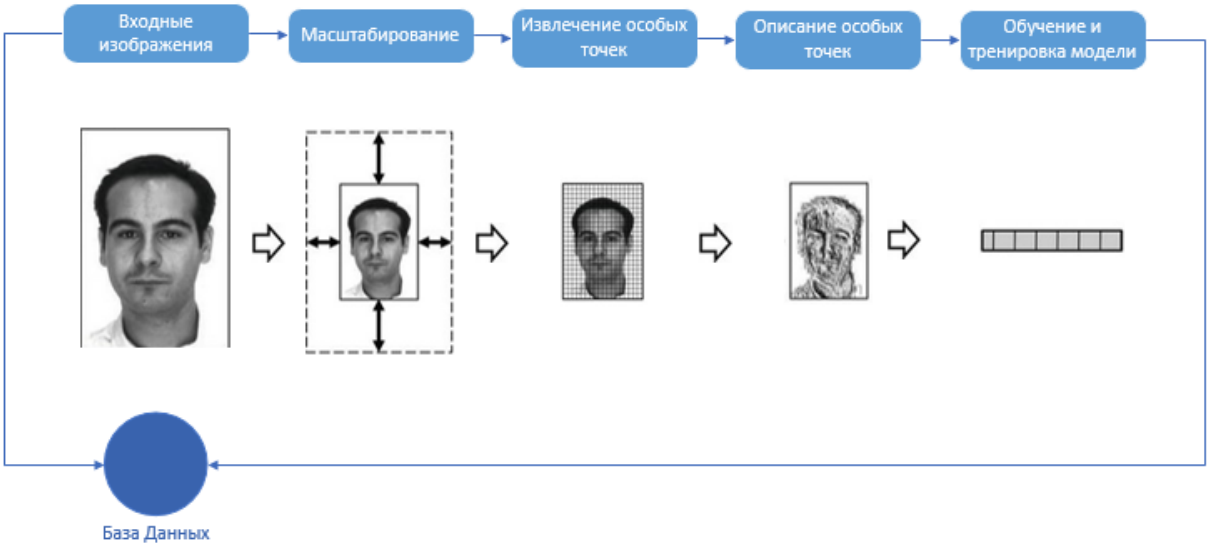

Рисунок 3 – Алгоритм работы системы распознавания лиц

Как показывает рисунок 3, в конце работы дескриптора получается вектор, содержащий информацию о местоположении и свойствах особых точек. На основе этих векторов строится модель нейронной сети для распознавания лиц.

После формулирования алгоритма работы систем распознавания лиц необходимо рассмотреть основные способы сравнивания дескрипторов, чтобы на их основе создать более эффективный алгоритм.

Для создания оптимального алгоритма сравнения дескрипторов особых точек был проведен анализ уже существующих решений.

Для оценки эффективности дескриптора авторы из [6] предлагают рассчитывать среднее время расчёта дескриптора для множества особых точек на изображении.

Для эксперимента формируется база изображений, состоящая из 5-10 изображений. Одно из изображений в этой базе является эталонным, другие являются искажёнными дубликатами исходного изображения.

Пример тестовой базы изображений для эксперимента представлен на рисунке 4.

Рисунок 4 – Пример тестовой базы изображений

Также предлагается использовать тестовые наборы с разными изображениями, на которых изображены однотипные объекты (здания, машины, лица, пейзажи). Дескрипторам необходимо определить конкретный тип объекта на каждом из изображений в наборе. Таким образом можно определить и сравнить точность дескрипторов [13].

Пример наборов изображений с однотипными объектами представлен на рисунке 5.

Рисунок 5 – Пример наборов изображений с однотипными объектами

В [6] приводится сравнение дескрипторов SURF, SIFT, BRISK, AKAZE, ORB, FREAK, DAISY, BRIEF и LATCH. Это наиболее актуальный набор дескрипторов среди изученных источников, однако эффективность дескрипторов сравнивается только на небольших наборах изображений. Необходимо увеличить набор исходных изображений до нескольких сотен или тысяч для получения более объективных показателей [12].

Пример одного из результатов эффективности дескрипторов из статьи на основе набора изображений стены представлен на рисунке 6.

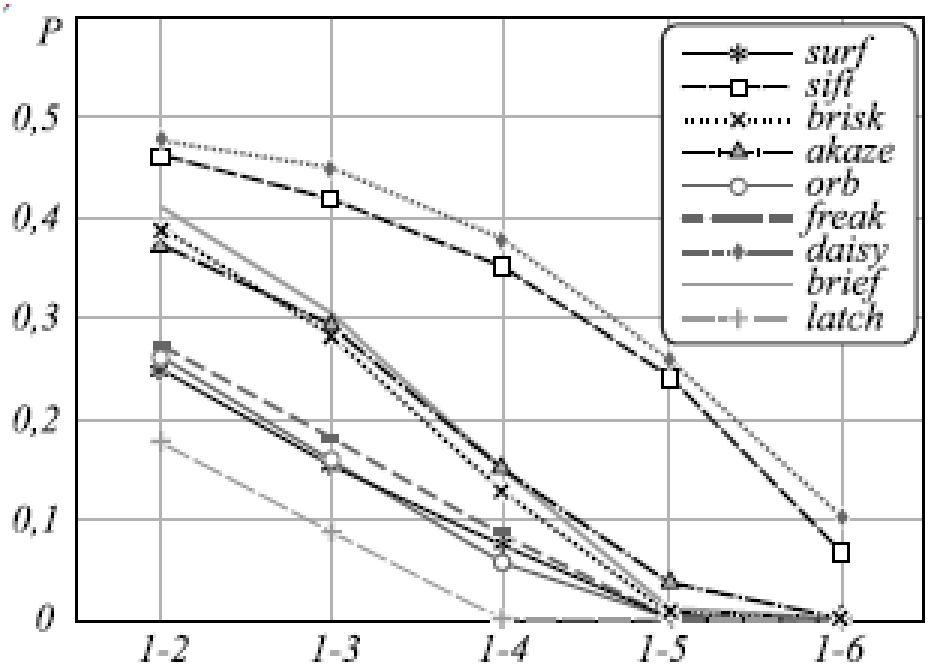

Рисунок 6 – Пример результата, полученного авторами статьи

Маленький объем выборки приводит к неточности в выходных данных. Как видно из рисунка 6, эффективность многих дескрипторов совпадает на определённых отрезках. Это связано с большой погрешностью в исследованиях.

В работе [14] также рассматривается эффективность и быстродействие дескрипторов особых точек SURF, BRISK, FREAK и Block. Автор работы сравнивает среднее количество найденных особых точек каждого из дескрипторов. Также находится процентное соотношение найденных особых точек к их общему числу. Результаты представлены на рисунке 7.

SURF	BRISK	FREAK	Block
Пейзаж 266/267 99% Портрет 30/35 86% Документ 1000/1200 83%	Пейзаж 182/182 100% Портрет 15/29 50% Документ 1450/1456 99%	Пейзаж 157/157 100% Портрет 3/4 75% Документ 228/234 97%	Пейзаж 140/172 81% Портрет 12/36 33% Документ 357/562 64%
Пейзаж 24/25 96% Портрет 2/9 22% Документ 140/186 75%	Пейзаж 21/21 100% Портрет 1/16 6% Документ 97/100 97%	Пейзаж 18/18 100% Портрет 0/6 0% Документ 57/66 83%	Пейзаж 35/43 81% Портрет 0/4 0% Документ 350/543 64%
Пейзаж 343/346 99% Портрет 5/16 31% Документ 600/782 76%	Пейзаж 330/330 100% Портрет 5/5 100% **Документ** **1290/1298 99%**	Пейзаж 211/211 100% Портрет 0/3 0% Документ 200/227 88%	Пейзаж 383/386 98% Портрет 7/7 100% Документ 800/1043 80%
Пейзаж 152/152 100% Портрет 0/3 0% Документ 800/1284 62%	Пейзаж 127/127 100% Портрет 0/0 0% Документ 1600/1665 96%	Пейзаж 79/79 100% Портрет 1/7 14% Документ 500/556 90%	Пейзаж 200/203 98% Портрет 2/4 50% Документ 1200/1500 80%
Пейзаж 146/146 100% **Портрет** **38/46 84%** Документ 100/126 79%	Пейзаж 65/65 100% Портрет 0/0 0% Документ 68/72 94%	Пейзаж 54/54 100% Портрет 3/4 75% Документ 23/27 85%	Пейзаж 72/80 90% Портрет 1/2 50% Документ 200/272 74%
Пейзаж 894/905 98% Портрет 111/172 65% Документ 500/939 53%	**Пейзаж** **835/835 100%** Портрет 36/40 90% Документ 270/297 90%	Пейзаж 392/394 99% Портрет 3/3 100% Документ 240/297 80%	Пейзаж 700/767 91% Портрет 14/15 93% Документ 600/1151 52%

Рисунок 7 – Результаты, полученные автором статьи [14]

На основе этих данных автор определяет наиболее эффективные дескрипторы для трёх типов изображений: пейзаж, портрет и документы.

Данный метод имеет несколько недостатков. Во-первых, собрано слишком мало данных по некоторым сочетаниям дескрипторов и изображений. Например, среднее количество особых точек, найденных методом SURF на портретных изображениях равно 5 и слишком мало для адекватной оценки. Во-вторых, общее количество особых точек на портретных изображениях, взятых автором, колеблется от 0 до 200. Этого недостаточно для заключения объективных выводов по работе дескрипторов.

Таким образом, оба метода нуждаются в доработке и улучшении. Предлагается брать большие объемы выборок, рассчитывая точность (процентное соотношение найденных особых точек к их общему числу) и скорость работы (время обработки набора изображений) дескрипторов, а также реализовать алгоритм сравнения в виде программного обеспечения.

1.2 Постановка задачи

Исходя из проанализированных данных и работ были сформулированы следующие задачи:

– изучение и описание математической модели дескриптора SIFT,
– изучение и описание математической модели дескриптора AKAZE,
– создание математической модели для сравнения изученных дескрипторов,
– реализация математической модели в виде программного обеспечения.

На рисунке 8 показана UML - диаграмма модели для сравнения дескрипторов лиц.

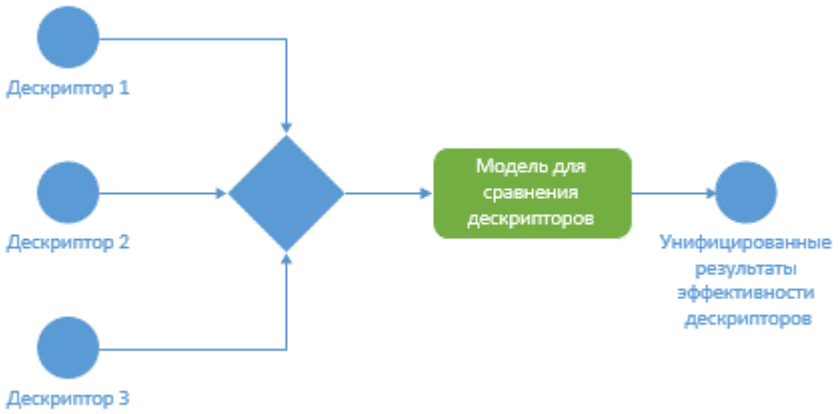

Рисунок 8 – UML - диаграмма модели для сравнения дескрипторов

Также была разработана диаграмма взаимодействия пользователя с программным обеспечением. UML - диаграмма принципа работы программного обеспечения показана на рисунке 9.

Рисунок 9 – UML - диаграмма принципа работы программного обеспечения

На основе вышеописанной диаграммы необходимо разработать алгоритм для объективного сравнения дескрипторов особых точек и подсчета их эффективности.

2 Разработка математической модели для сравнения дескрипторов

2.1 Математическая модель дескриптора BRIEF

С приходом компьютерного зрения на мобильные платформы возросла актуальность быстрых и легких дескрипторов. Так как мобильные приложения, использующие дескрипторы для распознавания лиц, стараются завоевать максимально широкую аудиторию с разным техническим обеспечением, возникает вопрос о быстром вычислении дескрипторов при малом потреблении оперативной памяти смартфонов. Вопрос скорости и легковесности также актуален для настольных и веб-приложений, старающихся использовать ресурсы системы максимально эффективно.

BRIEF считается одним из наиболее быстрых дескрипторов, так как использует сокращенное представление вектора. Для этого применяются алгоритмы снижения размерности (Dimensionality Reduction). С другой стороны, из-за этого дескриптор обладает относительно низкой точностью распознавания. Похожего эффекта снижения размерности можно добиться применением хеш-функции к дескриптору SIFT для его преобразования к бинарному виду.

В то время как большинство других дескрипторов используют снижение размерности к уже вычисленному дескриптору, BRIEF получает сокращенное представление вектора непосредственно при расчете масок.

Алгоритм использует маски размером 3x3 пикселя. На первом этапе все «соседи» центральной точки сравниваются со значением выделенного пикселя. На втором этапе те «соседи», значения которых меньше значения в центре, обозначаются нулями, а остальные – единицами. Это можно вывести в виде формулы 1.

$$\tau(p,x,y)=\begin{cases}1, p(x)<p(y)\\0, p(x)\geq p(y)\end{cases} \quad (1)$$

где p(x) – интенсивность пикселя в точке с координатами x = (u, v) в конкретной маске.

Далее каждый соседний пиксель с единицей возводится в степень 2^p, где p – номер позиции пикселя от 0 до 7. Таким образом, BRIEF дескриптор особой точки определяется как nd-мерная битовая строка, определяемая по формуле 2:

$$f_{nd=} \sum 2^{i-1} \tau(p, x, y) \quad (2)$$

На последнем этапе значения всех соседних пикселей суммируются в центральном пикселе. Алгоритм работы этих этапов представлен на рисунке 10.

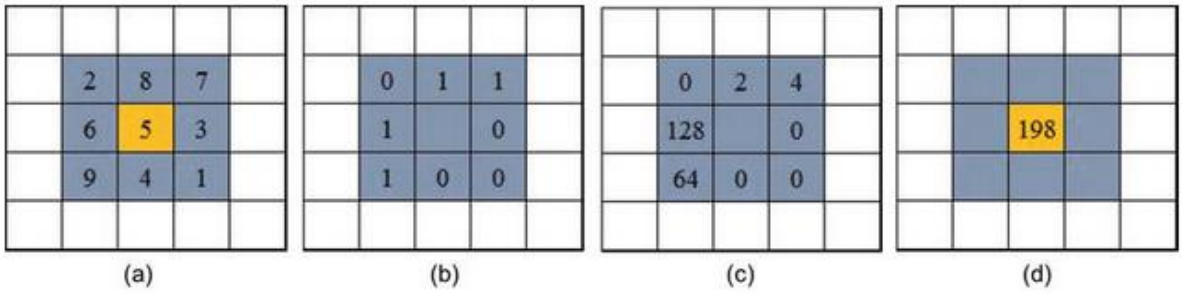

Рисунок 10 – Применение маски на участок изображения

Данный алгоритм повторяется для всех масок на изображении. Как показывает рисунок 11, в ходе работы алгоритма маски применяются ко всему изображению, выдавая на выходе обработанную матрицу.

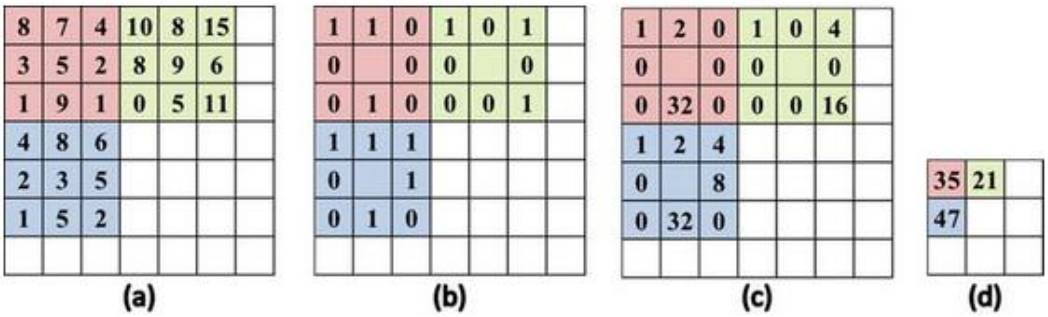

Рисунок 11 – Применение масок на изображение

Однако это упрощенный вариант и на практике алгоритм работает с использованием масок большего размера S. Это означает, что сравнивать точку в центре с соседними не получится, так как маска будет больше размера 3 на 3. Необходим специальный подход к выбору точек и получение результата теста $\tau(p, x, y)$. Поэтому одним из этапов является выбор участков для генерации тестов на маске. Всего существует 5 способов подбора тестовых пар в пространстве S x S. Они представлены на рисунке 12.

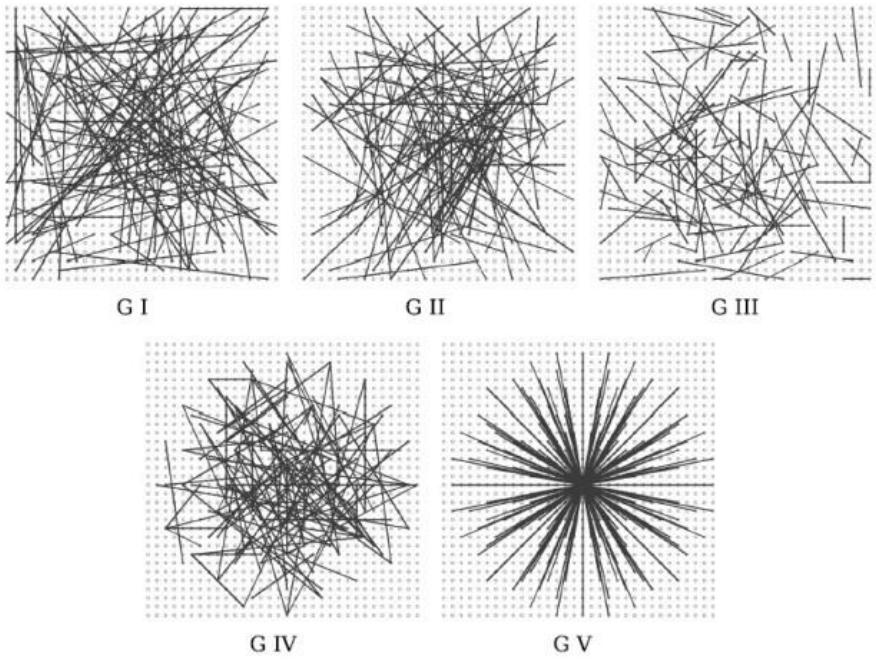

Рисунок 12 – Методы определения масок на изображении

– универсальное распределение $(-\frac{S}{2}, \frac{S}{2})$ для обеих точек. Точки равномерно распределены по маске и могут находится близко к её границам;
– нормальное распределение Гаусса $(0, \frac{1}{25}S^2)$ для обеих точек. Тесты генерируются на основе изотропного распределения Гаусса;
– нормальное распределение Гаусса $(0, \frac{1}{25}S^2)$ для первой точки. Вторая точка выбирается по распределению Гаусса $(x_i, \frac{1}{100}S^2)$ с центром во второй точке. Таким образом тест становится более «локальным». Если вторая точка выходит за границы маски, то она остаётся на самой границе;
– точки выбираются случайным образом из дискретных местоположений на грубой полярной сетке, созданной при помощи квантования;
– первая точка находится в центре маски, а вторая принимает все возможные значения на грубой полярной сетке.

Для каждого из описанных методов была рассчитана точность распознавания на тестовых изображениях. Результаты представлены на рисунке 13.

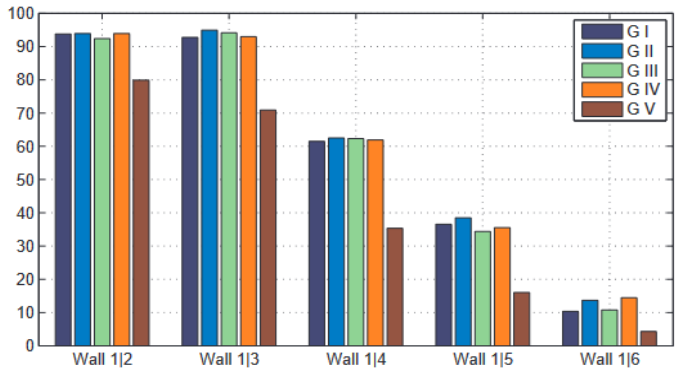

Рисунок 13 – Точность распознавания каждого из методов определения масок

Как видно из рисунка 13, 5-ый метод обладает худшей точностью из всех представленных. Самой высокой точностью обладает 2-ой метод подбора масок. Соответственно, при реализации дескриптора будет использоваться именно он [24].

К сожалению, дескриптор обладает низкой устойчивостью к масштабированию и вращению изображения. Как показывают результаты [15], BRIEF сильно уступает SIFT и SURF по этим параметрам [17,22]. Таким образом, дескриптор является интересным кандидатом для сравнения в условиях распознавания лиц, обладая ярко выраженными преимуществами и недостатками.

2.2 Математическая модель дескриптора SIFT

Масштаб изображения – одно из главных свойств обрабатываемого изображения. В зависимости от масштаба изменяется восприятие объекта и его ключевых точек дескриптором. Например, если человек смотрит на дерево издалека или на молекулярном уровне, то он не увидит веток и листьев на нём. Они станут значимы и превратятся в отдельные объекты только в определенном масштабе. Также и для алгоритмов, распознаваемый объект может просто не обрабатываться дескриптором при определённом масштабе.

Одним из главных преимуществ дескриптора SIFT является его устойчивость к масштабированию изображения [25, 27]. Это значительно повышает эффективность распознавания объектов, но в области распознавания лиц этот фактор не столь важен по двум причинам. Во-первых, лица практически не имеют резких углов, обладая гладкими очертаниями. Во-вторых, системы распознавания лиц, как правило, обучаются и обрабатывают изображения в одинаковом масштабе. Тем не менее устойчивость к изменению масштаба изображения сокращает количество артефактов на конечном изображении, улучшая точность распознавания объектов [28].

Рисунок 14 показывает побочный эффект масштабирования изображения – при увеличении угол превращается в грань. Это может повлиять на работу дескриптора, который весьма вероятно пропустит эту ключевую точку.

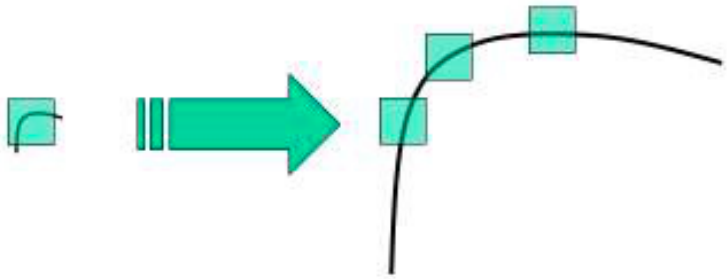

Рисунок 14 – Превращение угла в грань при увеличении масштаба

Каким же образом SIFT добивается устойчивости к масштабированию? В основе дескриптора лежит алгоритм разности Гауссианов (Difference of Gaussian), который находит особые точки на разных масштабах одного изображения. На рисунке 15 показано пирамидальное представление разных масштабов изображения.

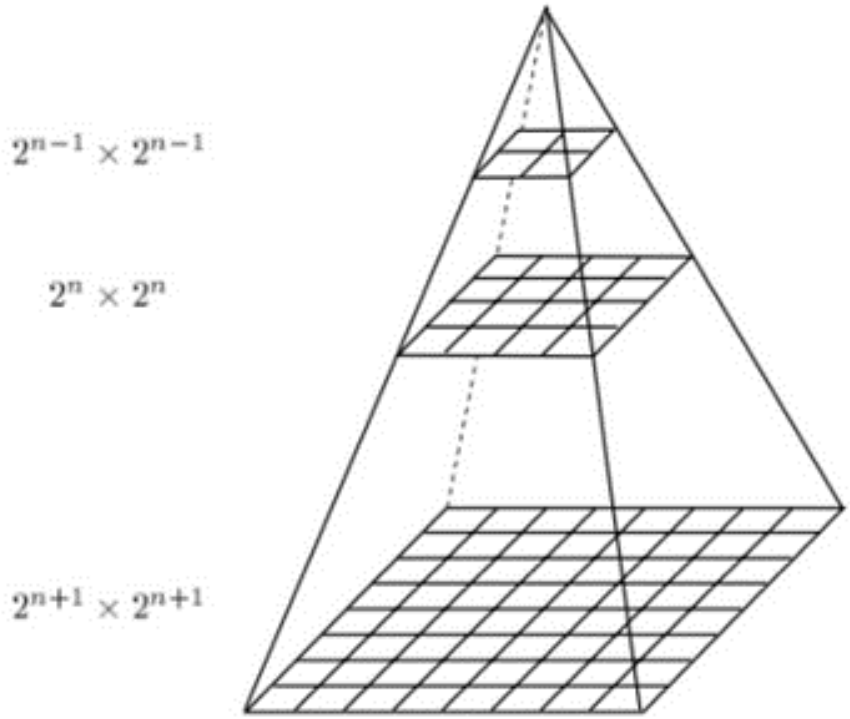

Рисунок 15 – Пирамидальное представление разных масштабов

Сначала исходное изображение слегка размывается с помощью Гауссова ядра. Затем подсчитывается сумма производных 2-го порядка. На основе этой суммы находятся углы и грани изображения, которые могут являться ключевыми точками. Фильтр Гаусса соответствует следующей формуле:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad (3)$$

где L(x,y,σ) – это значение гауссиана в точке с координатами (x,y) и радиусом размытия σ;

$I(x, y)$ – значение исходного изображения,

$G(x, y, \sigma)$– гауссово ядро, рассчитываемое по формуле 4:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad (4)$$

Далее рассчитывается разность гауссиан. Из каждого пикселя изображения вычитается пиксель изображения с другой степенью размытия:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \quad (5)$$

На рисунке 16 представлен пример генерации пирамиды масштабов.

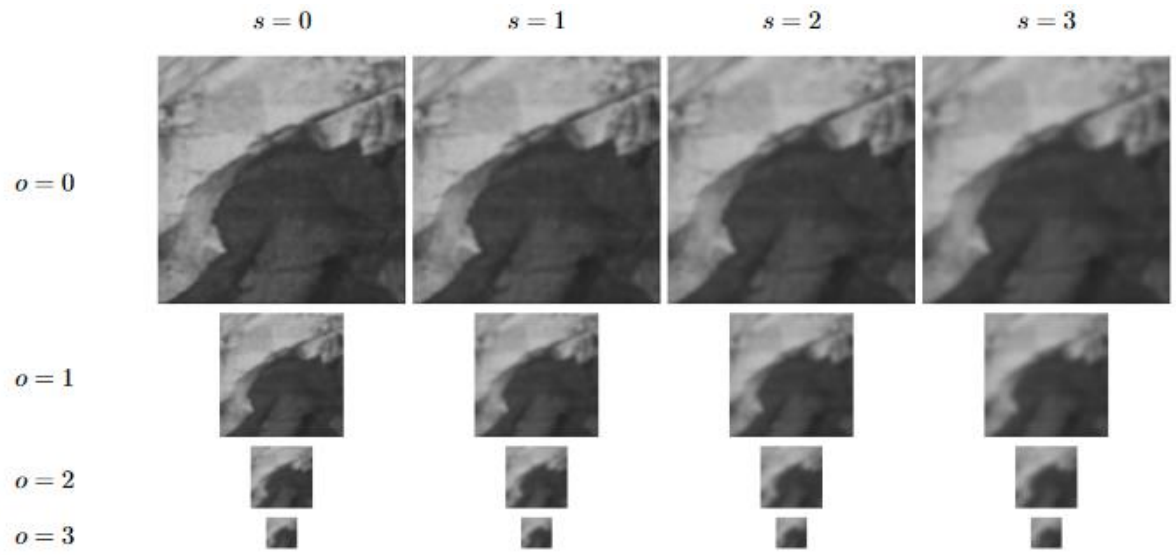

Рисунок 16 – Пример пирамиды масштабов, сгенерированной из исходного изображения

Размытие увеличивается слева направо, а октавы увеличиваются сверху вниз. Каждая следующая октава получается путём понижения степени двойки для количества пикселей изображения.

В то же время будет строиться пирамида разностей гауссианов, в которой будет на одно изображение меньше, чем в пирамиде гауссианов. Создание пирамид представлено на рисунке 17.

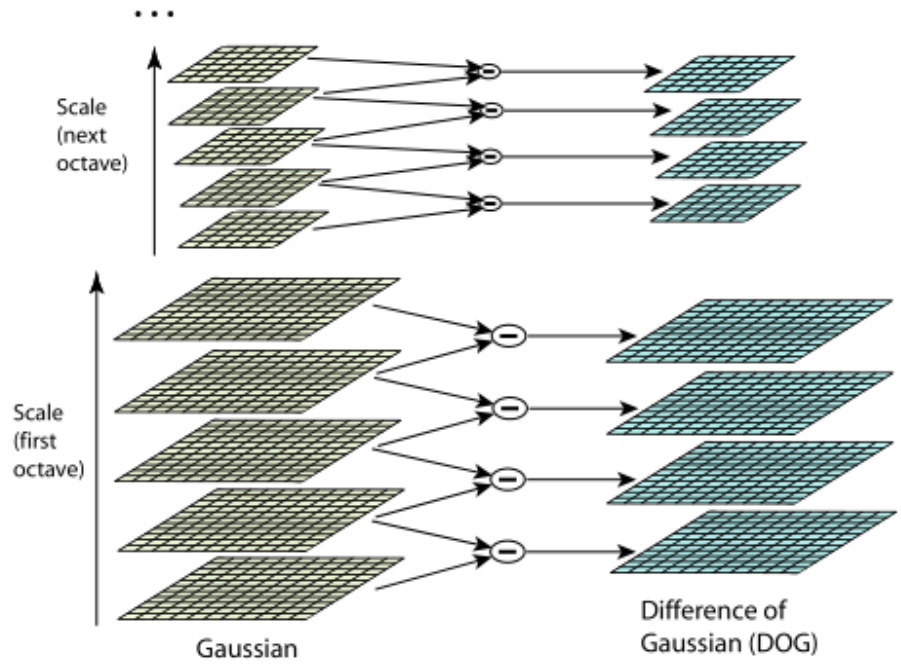

Рисунок 17 - Построение пирамиды разностей гауссианов

На рисунке 18 представлен пример создания пирамиды разностей гауссианов для изображения, приведённого чуть выше. Каждый слой пирамиды разностей строится из двух соседних слоев пирамиды гауссианов.

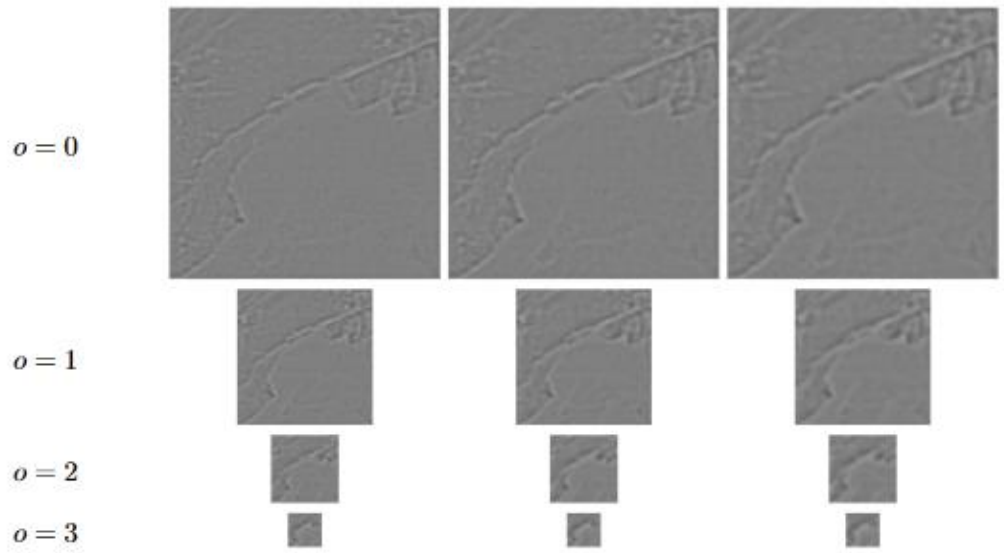

Рисунок 18 – Пример построение пирамиды разностей гауссианов

После обнаружения особых точек их нужно локализировать. Второй этап алгоритма SIFT определяет расположение ключевых точек до субпиксельной точности, параллельно удаляя любые ложные особые точки. Если значение в конкретной точке существенно отличается от значений точек с кругом, то такая точка будет являться локальным экстремумом разности гауссианов, то есть считаться особенной. Пример особой точки представлен на рисунке 19. Экстремум помечен крестиком и является особой точкой. Стоит отметить, что экстремум является максимум или минимумом среди трёх измерений x,y,σ.

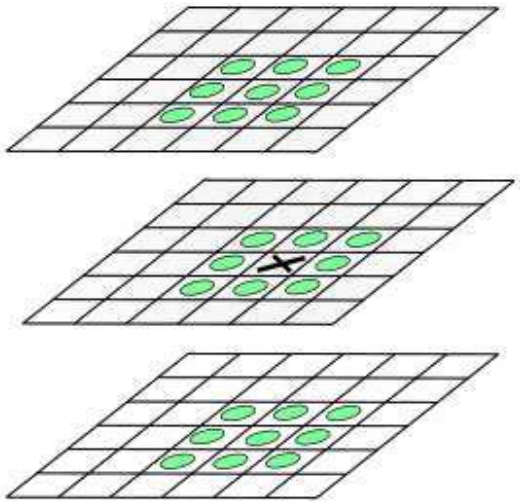

Рисунок 19 - Определение особой точки по локальному экстремуму

Третьим шагом алгоритма является присвоение ориентации особой точке. Устойчивость к вращению изображения достигается путём определения направления для каждой найденной особой точки.

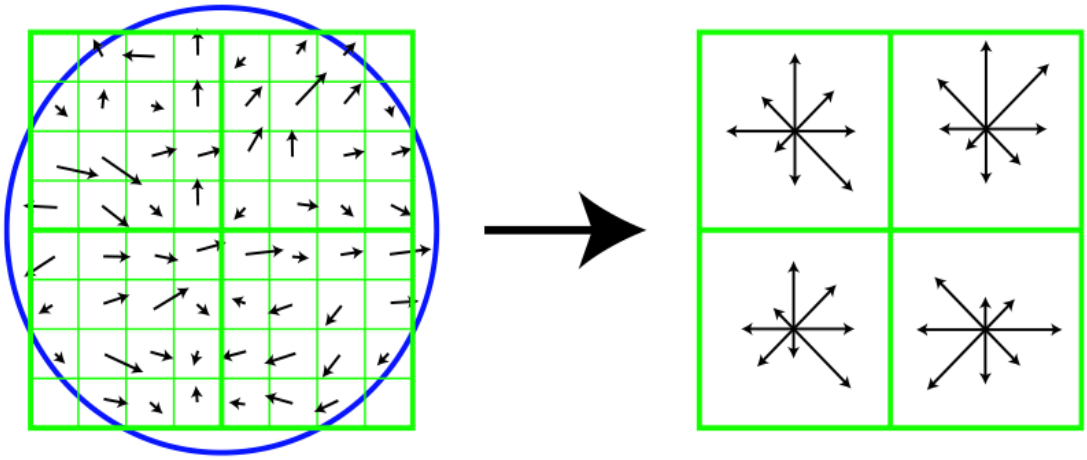

Рисунок 20 – Определение направления особой точки

На рисунке 20 показано визуальное построение гистограммы ориентации. Ориентация изображения определяет какое направление на гистограмме должно быть указано для каждого пикселя. Значение, указанное в каждой из ячеек, задаётся величиной градиента, рассчитанное с помощью оконной функции Гаусса с центром в особой точке.

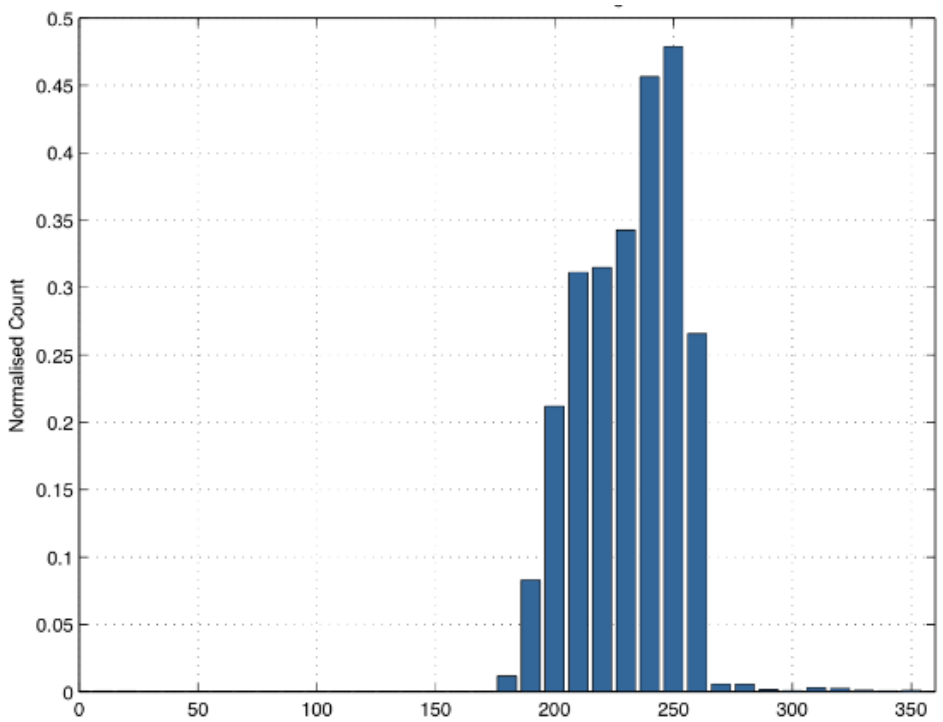

Рисунок 21 – Гистограмма ориентации

Как видно из рисунка 21, для каждой особой точки строится гистограмма ориентации, имеющая 36 делений, которые покрывают всю 360-градусную область. К пикам гистограммы используется полиномиальная регрессия для определения ориентации особой точки. Если гистограмма имеет более одного отличного пика, тогда создается несколько копий особой точки, каждая с одной из возможных ориентаций.

В конце концов, происходит генерация самого дескриптора. Дескриптор - это 128 - битный нормализованный вектор. На данном этапе уже имеется информация о направлении, местоположении и масштабе особых точек. По факту, дескриптор представляет собой ту же гистограмму ориентации, конвертированную в вектор.

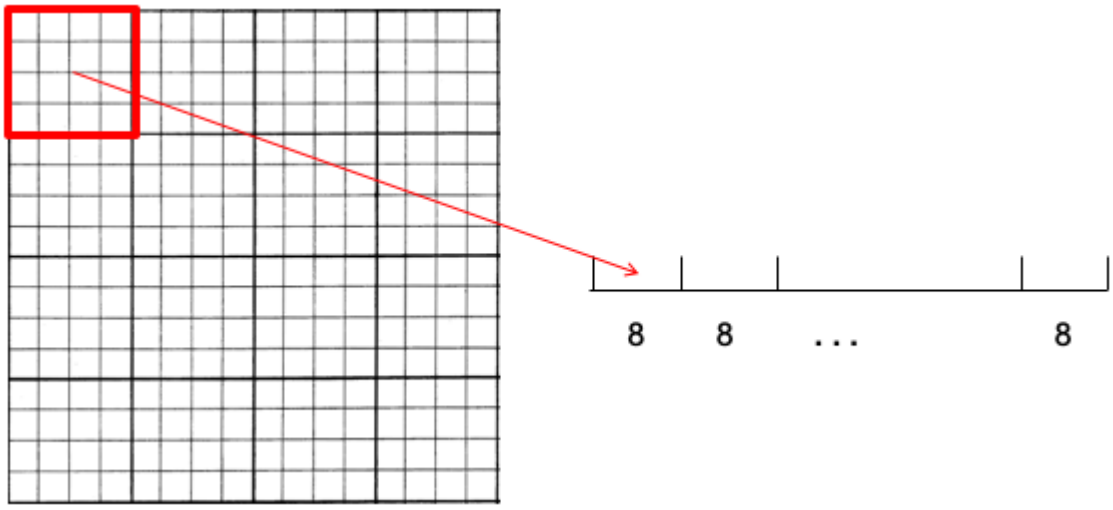

Рисунок 22 – Сетка гистограммы ориентации

Как видно из рисунка 22, изображение делится на области размером 4x4 клетки (размер сетки зависит от функции). Сетка центрируется по особой точке. Для каждой из 16 областей рассчитывается собственная гистограмма ориентации. Далее из ориентаций всех областей выбираются наиболее подходящие сегменты окружности с наибольшим количеством совпадающих направлений.

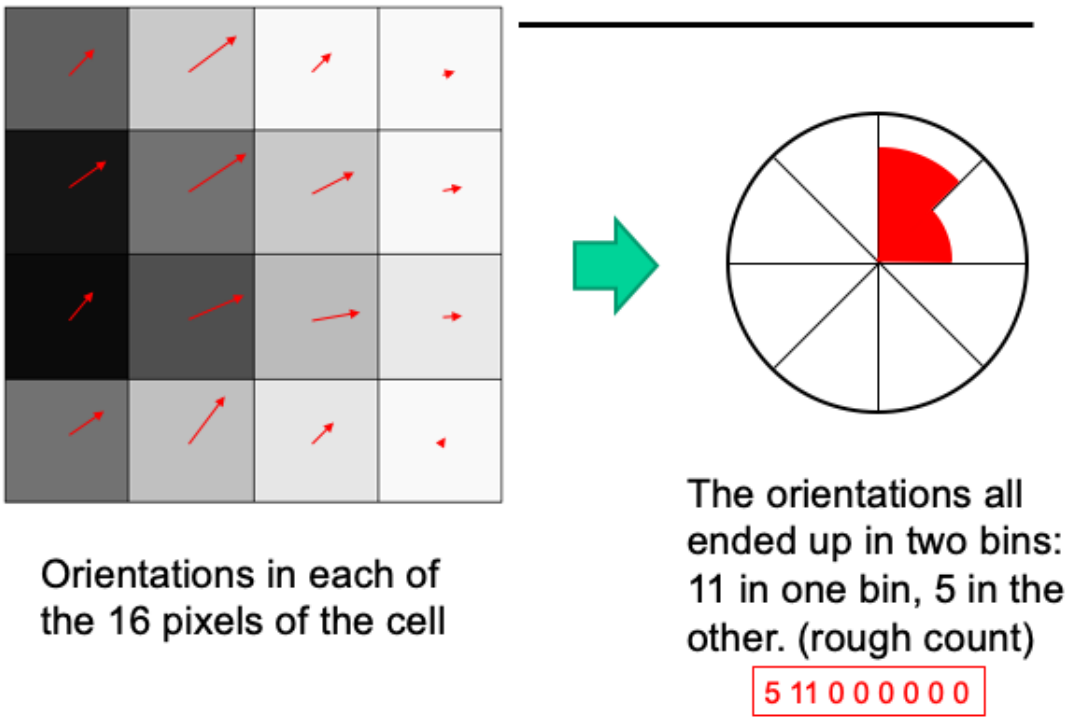

Рисунок 23 – Конвертация ориентации из областей в сегменты

Финальный этап представлен на рисунке 23. Полученный вектор изменяется, чтобы уменьшить эффект изменения освещения. Сначала вектор нормализуется до единичной длины. При изменении контраста изображения значение каждого пикселя умножится на определенную константу, поэтому нормализации вектора поможет избежать ошибок при разном контрасте. При изменении яркости изображения к значению каждого пикселя добавиться константа, но это также не затронет значения градиентов, которые рассчитываются через разность пикселей. Следовательно, дескриптор SIFT устойчив к изменению освещения. Однако нелинейное изменение освещения, вызванное сменой положения камеры, может привести к значительному изменению значений градиентов, но не затронет их ориентацию.

Таким образом, была рассмотрена математическая модель дескриптора SIFT и его алгоритм работы.

2.3 Математическая модель дескриптора AKAZE

Первым этапом работы дескриптора AKAZE является вычисление нелинейного пространства масштаба (scale space). AKAZE использует подход дескриптора SIFT, дискретизируя по логарифмическим шагам пространство масштаба. Однако в отличие от SIFT, AKAZE работает с изображением в исходном разрешении без понижения дискретизации на следующих октавах.

Набор октав и подуровней представляется дискретным октавным индексом o и подуровнем s. Индексы октавы и подуровня распределяются по соответствующим масштабам σ по формуле 6.

$$\sigma_i(o,s) = \sigma_0 2^{o+\frac{s}{S}}, o \in [0 \dots O-1], s \in [0 \dots S-1], i \in [0 \dots N] \quad (6)$$

где σ_0- базовый уровень масштаба,

N - общее количество отфильтрованных изображений.

Далее набор дискретных уровней масштаба в пиксельных единицах σ_i преобразовывается в единицы времени. Причина этого преобразования заключается в том, что нелинейная диффузионная фильтрация определяется во времени. В случае с гауссовским пространством масштаба свертка изображения с гауссовым ядром среднеквадратичного отклонения σ (в пикселях) эквивалентно фильтрации изображения в течение некоторого времени t = $\sigma^2/2$. Применяя это преобразование, получается набор времён эволюции, который преобразовывает пространство масштаба $\sigma_i(o,s)$ в единицы времени с помощью следующего отображения $\sigma_i \rightarrow t_i$:

$$t_i = \frac{1}{2}\sigma_i^2, i \in \{0 \dots N\} \quad (7)$$

Стоит отметить, что отображение $\sigma_i \rightarrow t_i$ используется только для получения набора времен эволюции, из которого мы строим нелинейное

пространство масштаба. В общем, нелинейное пространство масштаба на каждом отфильтрованном изображении t_i результирующего изображения не соответствует свертке исходного изображения с гауссианом среднеквадратичного отклонения σ_i. Однако данная структура совместима с гауссовским пространством масштаба [26].

Сначала изображение сворачивается с гауссовым ядром среднеквадратичного отклонения σ_0 для уменьшения шума и возможных артефактов изображения. Пример шума на изображении представлен на рисунке 24.

Рисунок 24 - Набор изображений с увеличивающимся шумом

Из свёрнутого изображения вычисляется гистограмма градиента изображения и получается параметр контраста. Далее на основе данного параметра контраста и набора этапов эволюции t_i строится нелинейное пространство масштаба итеративным путём с использованием формулы 8:

$$L^{i+1} = \left(I - (t_{i+1} - t_i) * \sum_{l=1}^{m} A_l\left(L^i\right)\right)^{-1} L^i \quad (8)$$

На рисунке 25 показаны сравнение гауссова и нелинейного пространства масштаба на нескольких этапах эволюции t_i. Как можно заметить из рисунка 25, гауссово размытие сглаживает все структуры на изображении, в то время как нелинейное размытие оставляет края объектов нетронутыми.

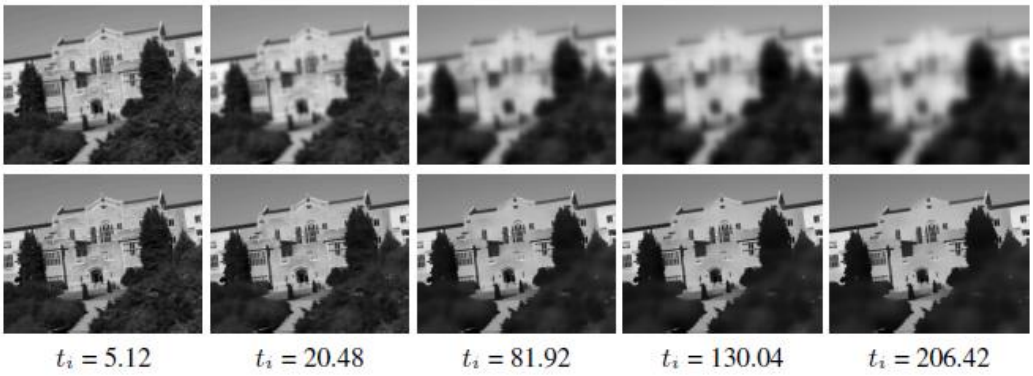

Рисунок 25 - Сопоставление гауссова и нелинейного пространства масштаба на нескольких этапах эволюции t_i

Вторым шагом работы дескриптора является обнаружение особых точек на изображении. Для обнаружения особых точек вычисляется отклик нормированных по масштабу детерминантов гауссиан на нескольких уровнях масштаба [27]. Для обнаружения многомерных особых точек набор дифференциальных операторов необходимо нормировать относительно масштаба. В то же время, амплитуда производных x и y убывает по формуле 9:

$$L_{Hessian} = \sigma^2\left(L_{xx}L_{yy} - L_{xy}^2\right) \quad (9)$$

где (L_{xx}, L_{yy}) - производные второго порядка по горизонтали и вертикали соответственно,

L_{xy} - смешанная производная второго порядка.

Учитывая набор отфильтрованных изображений из нелинейного пространства масштаба L^i, анализируется отклик детектора на разных уровнях масштаба σ_i. Для этого находятся максимумы по масштабу и пространству.

Поиск экстремумов происходит во всех отфильтрованных изображениях за исключением первого и последнего в прямоугольной области размером σ_i × σ_i по текущему i, верхнему i + 1 и нижнему i - 1 отфильтрованному

изображению. Для ускорения поиска экстремумов сначала проверяются ответы в окне размером 3 × 3 пикселя, чтобы быстро отбросить все не максимальные ответы. В заключение, положение ключевой точки оценивается с субпиксельной точностью.

Набор производных первого и второго порядка аппроксимируется с помощью 3 × 3 фильтров Шарра с разными размерами шага производной σ_i. Производные второго порядка аппроксимируются с использованием последовательных фильтров Шарра в необходимых координатах производных. Этот фильтр аппроксимирует инвариантность вращения значительно лучше, чем другие популярные фильтры, например, фильтр Собеля или стандартное дифференцирование центральных разностей [2].

После обнаружения особых точек, дескриптору необходимо их описать. Сначала находится доминирующая ориентация особой точки. Для этого находятся производные первого порядка L_x и L_y, которые преобразуются в векторы. Самый длинного вектор будет являться доминирующей ориентацией.

Финальным этапом работы дескриптора является его построение. Для обнаруженных особых точек в масштабе σ_i вычисляются производные первого порядка L_x и L_y размера σ_i на прямоугольной сетке $24\sigma_i \times 24\sigma_i$. Эта сетка делится на 4 × 4 подобласти размером $9\sigma_i \times 9\sigma_i$ с перекрытием $2\sigma_i$.

Полученные производные взвешиваются гауссианом ($\sigma_i = 2.5\sigma_i$) с центром в особой точке и суммируются в вектор дескриптора $d_v = (\sum L_x, \sum L_y, \sum|L_x|, \sum|L_y|)$. Затем каждый вектор подобласти взвешивается с помощью гауссиана ($\sigma_2 = 1.5\sigma_i$), определенного на маске 4 × 4 с центром в особой точке.

Все отсчёты в прямоугольной сетке поворачиваются согласно ранее найденной доминирующей ориентации. Это позволяет дескриптору быть устойчивым к вращению изображения. Наконец, вектор дескриптора длиной 64 символа нормализуется в единичный вектор для достижения инвариантности к контрасту.

Для подтверждения эффективности дескриптора AKAZE перед его предшественником KAZE, был проведено исследование их эффективности на разных наборах изображений.

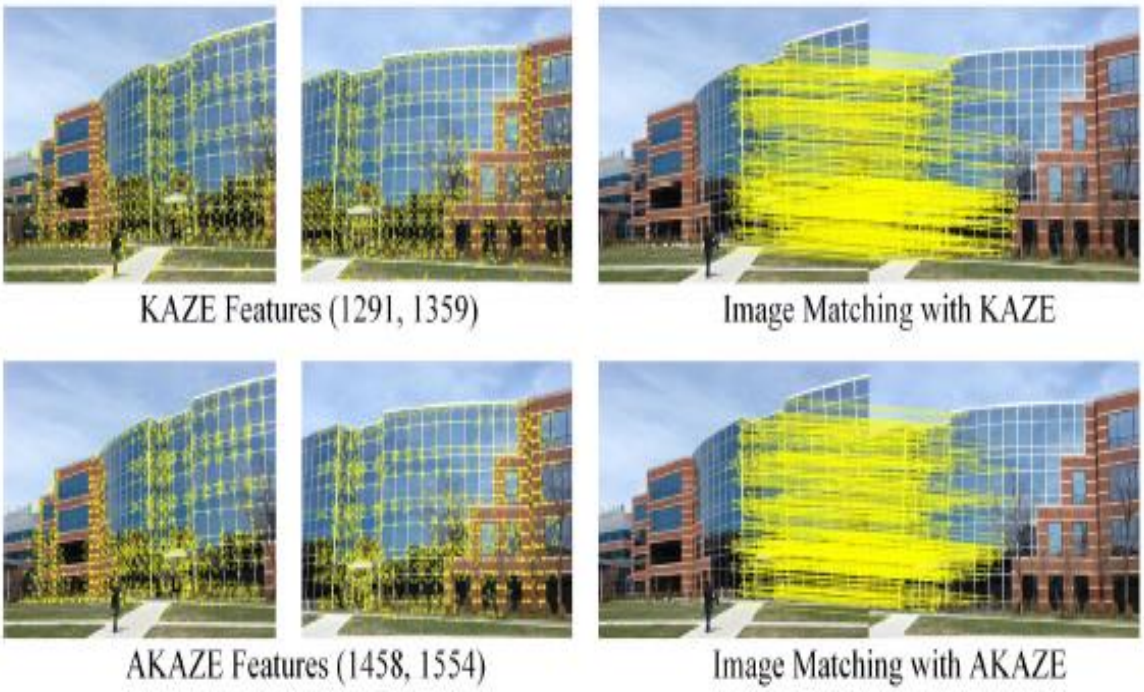

Рисунок 26 - Сравнение дескрипторов AKAZE и KAZE

Как видно из рисунка 26, AKAZE нашёл 1458 и 1554 особых точек на первом и втором изображениях соответственно, в то время как KAZE определил только 1291 и 1359 особых точек на двух изображениях. Это на 13% меньше результата AKAZE. Также AKAZE нашёл больше соответствий между этими изображениями, что подтверждает его эффективность на практике.

Таким образом, были рассмотрены основные достоинства и недостатки дескриптора AKAZE, а также рассмотрена его математическая модель.

2.4 Математическая модель сравнения дескрипторов

Изучив принципы работы различных дескрипторов особых точек, необходимо составить собственную модель для корректного сравнения дескрипторов. Модель должна делать объективные выводы из результатов работы различных дескрипторов, поэтому необходимо использовать универсальные параметры, которые можно извлечь из каждого дескриптора.

Основными параметрами модели можно считать:

– время обработки одного изображения t. Измерения данного параметра необходимо производить на большой выборке данных (многократный запуск), иначе результаты будут не объективными;

– чувствительность r (recall) – отношение верно найденных особых точек к числу всех найденных точек. Дескрипторы могут найти ложные особые точки, которые уменьшают качество распознавания лица. Как правило, на это влияет низкая освещённость и окклюзии изображения. Вычисляется по формуле 10. Количество правильных и неправильных особых точек находится сравнением обработанного дескриптором изображения с «эталонным» изображением. Для этого существуют специальные алгоритмы сопоставления (matchers), реализованные в библиотеке OpenCV;

$$r = \frac{\text{найденные точки}}{\text{правильные точки}} \tag{10}$$

– точность распознавания p (precision), рассчитываемая по формуле 11. На первый взгляд, этот показатель аналогичен чувствительности, однако на деле он дополняет его, позволяя добиться более объективной оценки дескрипторов;

$$p = 1 - \frac{\text{неправильные точки}}{\text{найденные точки}} \tag{11}$$

– тип дескриптора type, определяющий вид дескриптора особых точек, используемый при обработке изображения;
– порог фильтрации σ определяет, насколько чувствителен будет фильтр Гаусса к потенциальным особым точкам. Чем меньше это значение, тем больше особых точек будет на изображении и тем менее «качественными» они будут. Используется на входе модели и применяется на первом этапе построения дескрипторов;
– количество октав nOctaves определяет количество октав в используемом дескрипторе. Используется при инициализации модели и применяется на первом этапе построения дескрипторов;
– количество слоев октав nOctaveLayers определяет аналогичный параметр в каждом из дескрипторов. Используется при инициализации модели и применяется на первом этапе построения дескрипторов.

Также были рассмотрены следующие параметры, не включенные в модель:

– размер маски patchSize. Несмотря на то, что вычисление значений с использованием маски являются важным этапом работы в большинстве дескрипторов, введение размерности масок в модель может значительно усложнить сравнение, если дескриптор будет использовать маски, отличные от квадратных. Например, дескриптор DAISY использует круговые маски для вычисления дескриптора. Поэтому было принято решение оставить размерность масок на базовых значениях для сохранения объективности модели;
– в [16] рассматривается использование различных дополнительных параметров для вычисления дескрипторов. Одним из них является nFeatures или количество единовременно сохраняемых особых точек изображения. Он позволяет сократить количество особых точек, сохраняя лишь самые «важные» из них. Однако применение этого параметра может значительно повлиять на значение параметров

точности и чувствительности, поэтому было принято решение оставить этот параметр на базовом значении в 500 точек на изображение. Данное значение гарантирует сохранение всех особых точек при обработке изображений с лицами людей, так как в среднем такие изображения содержат не более 200 особых точек;

– порог контраста contrastThreshold отвечает за фильтрацию слабых особых точек на однородных областях изображения. Этот параметр может повысить качество распознавания лиц на слабо освещенных изображениях, однако при хорошем освещении он лишь ухудшит результаты выборки. Так как планируется брать наборы данных с разными типами освещенности, данный параметр не будет использоваться при построении модели.

В итоговом виде модель принимает следующий вид:

$$M(type, \sigma, nOctaves, nOctaveLayers) = I(t, r, p) \quad (12)$$

где $M(type, \sigma, nOctaves, nOctaveLayers)$ – исходное изображение,

type – тип дескриптора,

σ – порог фильтрации,

$nOctaves$ – количество октав,

$nOctaveLayers$ – количество слоёв октав,

$I(t, r, p)$ – обработанное изображение,

t – время обработки,

r – чувствительность дескриптора,

p – точность дескриптора.

3 Программная реализация математической модели для сравнительного анализа дескрипторов

3.1 Описание программного обеспечения

Для начала необходимо определиться с языком разработки приложения. Проведём сравнительный анализ основных языков программирования для разработки стационарных приложений:

– C#,

– C,

– C++,

– Java,

– Python,

– Delphi.

C - это язык программирования общего назначения высокого уровня, который в последние года получил новый приток популярности. C входит в число наиболее широко используемых языков, имеет компилятор для большинства компьютерных систем и повлиял на многие популярные языки программирования, например, C++ или C#.

C# (произносится как «Си шарп») - это объектно-ориентированный язык программирования от Microsoft, целью которого является объединение вычислительной мощности C ++ с простотой программирования Visual Basic. C# основан на C++ и содержит функции, аналогичные функциям Java. C# разработан для работы с платформой Microsoft .NET.

C++ - это язык объектно-ориентированного программирования (ООП), который многие считают лучшим языком для создания крупномасштабных приложений. C++ - это надстройка языка C.

Java является кроссплатформенным объектно-ориентированным языком программирования. Он обладает высокой производительностью при низком

пороге сложности. Является прямым конкурентом C++, обладая схожей областью применения и функциональностью.

Python - это интерпретируемый объектно-ориентированный язык программирования высокого уровня с динамической семантикой. Широкую популярность получил в начале 2010-ых годов, когда у программистов вновь возник интерес к изучению нейронных сетей и искусственного интеллекта. Имеет низкий порог вхождения благодаря простоте синтаксиса. Динамическая типизация делает его очень привлекательным для быстрой разработки приложений.

Delphi - это язык высокого уровня, поддерживающий объектно-ориентированный дизайн. Обеспечивает быструю разработка приложений, начиная от решений для баз данных и заканчивая мобильными приложениями, и используется как в Windows, так и в Linux.

Языки программирования будут оцениваться по следующим критериям:

- распространённость (10 баллов) – количество доступной информации для изучения по конкретному языку. Чем выше этот показатель, тем проще и быстрее можно найти ответ на ошибку компиляции или технический вопрос при разработке проекта. Баллы считаются на основе индекса TIOBE за 2021 год [29]: 1 место – 10 баллов, 2 место – 9 баллов и т.д. Данный индекс является одним из самых точных показателей популярности языка на основе анализа поисковых запросов;
- наличие опыта разработки (10 баллов) – количество опыта и знаний по конкретному языку программирования;
- производительность (10 баллов) – скорость выполнения машинного кода. Многие языки используют встроенные инструменты для оптимизации и сокращения кода, которые в той или иной мере сказываются на производительность. Например, в java существует garbage collector, который очищает ненужные объекты из памяти с помощью специальных алгоритмов. Это упрощает код, позволяя не

задумываться об очистке различных объектов, но в то же время повышает время компиляции;

– кроссплатформенность (5 баллов) – возможность выполнять один и тот же код приложения на различных платформах (Windows, Unix, Mac, Android, IOS), а также удобство разработки приложения на разных ОС. За каждую ОС засчитывается один балл.

Сравнительный анализ языков программирования представлен в таблице 1.

Таблица 1 – Сравнение языков программирования

Критерии оценки	C#	C	C++	Java	Python	Delphi
Распространённость	6	9	7	8	10	2
Опыт разработки	1	3	8	10	2	5
Производительность	8	10	8	7	5	6
Кроссплатформенность	3	3	3	4	5	2
Итого	18	25	26	30	22	15

Как видно из таблицы 1, лучшим выбором является объектно-ориентированный язык Java. Теперь можно перейти к выбору средств и инструментов разработки.

Основными средами разработки на языке Java являются:

– Eclipse,

– IntelliJ IDEA,

– NetBeans,

– Microsoft Visual Studio Code.

Eclipse является одной из самых первых интегрированных сред разработки на Java. Обладая широким функционалом и высокой скоростью работой, эта среда разработки быстро получила признание в Java-сообществе. Тем не менее, в последние годы популярность этой интегрированной среды разработки значительно упала из-за громоздкого интерфейса и большой нагрузки на систему.

IntelliJ IDEA – интегрированная среда разработки, обладающая качественным интерфейсом и высокой производительностью. Разработанная в начале 2000-ых годов, она стала основным конкурентом Eclipse на целый десяток лет. Обладает отличной интеграцией с системами контроля версий, системами сборки проектов и многими популярными java-библиотеками. Имеет бесплатную community версию и платную enterprise версию с дополнительным функционалом.

NetBeans является бесплатной кроссплатформенной интегрированной средой разработки с открытым исходным кодом. Обладает высокой производительностью, но поддерживает версии java до 11 включительно.

Microsoft Visual Studio Code является универсальной средой разработки для множества языков программирования. Поддержка языков программирования осуществляется за счёт плагинов и расширений для основного пакета. Из-за этого возникает большое количество багов в мало популярных языках программирования, поддержка которых осуществляется за счёт расширений пользователей. Также в данной среде разработки отсутствует поддержка популярных библиотек для конкретных языков программирования.

Интегрированные среды разработки можно оценить по следующим критериям:

- распространённость (10 баллов) – количество людей, пользующихся и развивающих конкретную среду разработки. Чем выше этот параметр, тем легче найти ответ на вопрос или ошибку компиляции;
- производительность (10 баллов) – количество ресурсов, требуемых IDE для компиляции, а также скорость выполнения кода;
- опыт разработки (10 баллов) – наличие опыта разработки в конкретной среде;
- сопровождение (5 баллов) – своевременное выявление и исправление багов, отзывчивая техподдержка, наличие магазина с расширениями.

Сравнительный анализ интегрированных сред разработки представлен в таблице 2.

Таблица 2 – Сравнение интегрированных сред разработки

Критерии оценки	Eclipse	IntelliJ IDEA	NetBeans	Microsoft Visual Studio Code
Распространённость	8	9	7	5
Производительность	5	6	10	7
Опыт разработки	8	10	6	8
Сопровождение	4	5	2	1
Итого	25	30	25	21

Как видно из таблицы 2, лучшим выбором среди интегрированных средств разработки является IntelliJ IDEA. Несмотря на ограниченный функционал community version, предоставленных инструментов хватит для разработки полноценного приложения. Следующим этапом является сравнение доступных библиотек компьютерного зрения.

Рассмотрим основные библиотеки компьютерного зрения, доступные для java:

– JavaCV,

– OpenCV,

– BoofCV,

– Deeplearning4j.

JavaCV является адаптированной под java оболочкой для библиотеки OpenCV, разработанная и выпускаемая с открытым исходным кодом на github. Обладает отличной производительностью и простотой понимания методов. Однако частота выхода новых версий оставляет желать лучшего. Новые версии выходят раз в полгода и далеко не весь функционал OpenCV получается адаптировать на java.

OpenCV является наиболее распространенной библиотекой компьютерного зрения для большинства языков программирования. Активно развивается сообществом C++ и python разработчиков. Обновления с новыми

алгоритмами выходят каждые 2 месяца, усложняя своевременную адаптацию библиотеки на другие языки программирования [18].

BoofCV является библиотекой компьютерного зрения с открытым исходным кодом. Содержит набор функции для простых задач компьютерного зрения. Несмотря на небольшой функционал и сложность изучения, обладает высокой производительностью [20].

Deeplearning4j является эксклюзивным для java набором инструментов для машинного обучения и компьютерного зрения. Обладает широким набором функций для продвинутых задач компьютерного зрения и искусственного интеллекта.

В качестве основных критериев выбора будут выступать:

– документация (10 баллов) – количество и качество документации по методам и классам конкретной библиотеки;
– простота использования (10 баллов) – понятность предлагаемых методов и функций. Простой интерфейс взаимодействия с библиотекой и её функционалом;
– опыт разработки (10 баллов) – опыт работы с конкретной библиотекой;
– сопровождение (5 баллов) – своевременное обновление процедур и функций, долгосрочная поддержка библиотеки разработчиком, регулярные обновления.

Таблица 3 – Сравнение библиотек компьютерного зрения

Критерии оценки	JavaCV	OpenCV	BoofCV	Deeplearning4j
Документация	5	10	8	9
Простота использования	10	8	6	5
Опыт разработки	5	8	3	2
Сопровождение	1	3	4	4
Итого	21	29	21	20

Как видно из таблицы 3, лучшим выбором является классическая библиотека по работе с компьютерным зрением OpenCV. Несмотря на слабую

поддержку этой библиотеки языком Java, она является ведущей библиотекой для работы с компьютерным зрением на многих языках программирования [4,17].

Таким образом, был определен набор инструментов и библиотек для разработки приложения: объектно-ориентированный язык программирования Java, интегрированная среда разработки IntelliJ IDEA Community Version и библиотека для работы с компьютерным зрением OpenCV. В дополнение к этому в проект внедрена система контроля версий github для удобной разработки и поддержки продукта.

3.2 Реализация программного обеспечения

После выбора средств разработки необходимо продумать архитектуру приложения и возможные архитектурные паттерны, которые помогут упростить разработку. Можно выделить следующие компоненты программы:

- классы, отвечающие за реализацию дескрипторов особых точек;
- классы, отвечающие за визуальный интерфейс приложения;
- класс для обработки изображений;
- основной класс программы.

В итоге получается большое количество классов, которые тесно связаны друг с другом. Это может привести к запутанному и непонятному коду с высокой степенью связности. Для решения подобных задач существует паттерн «медиатор» или «посредник», который позволяет уменьшить связность кода и упростить его понимание [9]. На рисунке 27 представлена возможная связанность элементов интерфейса программы.

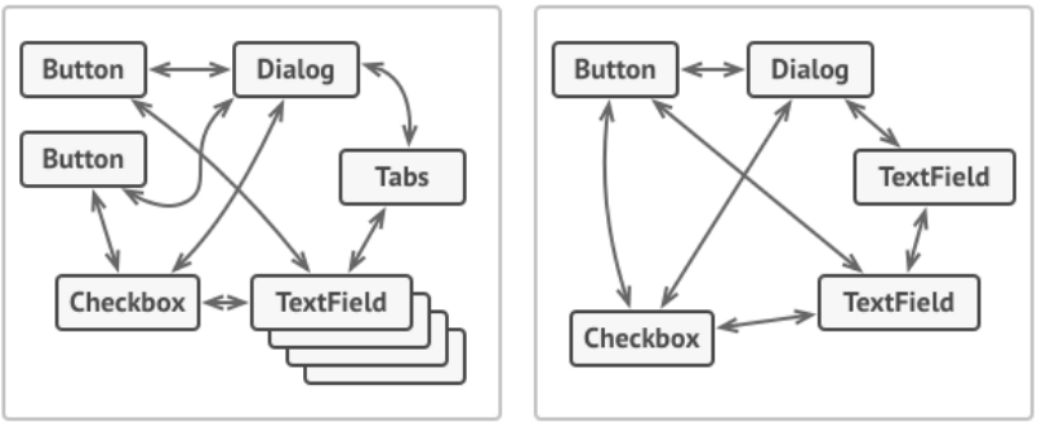

Рисунок 27 – Связи между классами до использования паттерна

Как видно из рисунка 27, каждый из классов имеет несколько связей с другими классами. Это приводит к повышенной сложности кода. Для решения проблемы вводится дополнительный класс, который принимает на себя роль связующего звена. На рисунке 28 этим классом является Dialog.

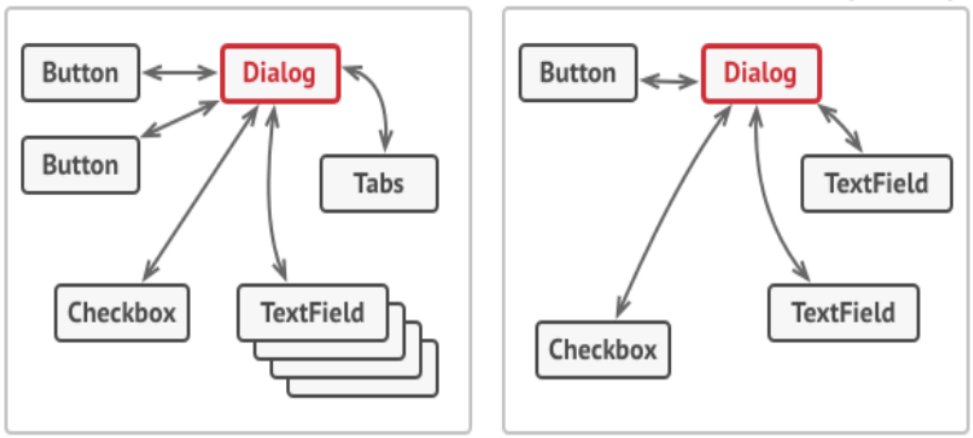

Рисунок 28 – Связи между классами после применения паттерна

Все методы контролируемых классов перенаправляются к классу посреднику, который их обрабатывает [21].

На рисунке 29 представлена структура паттерна «посредник» с пояснениями.

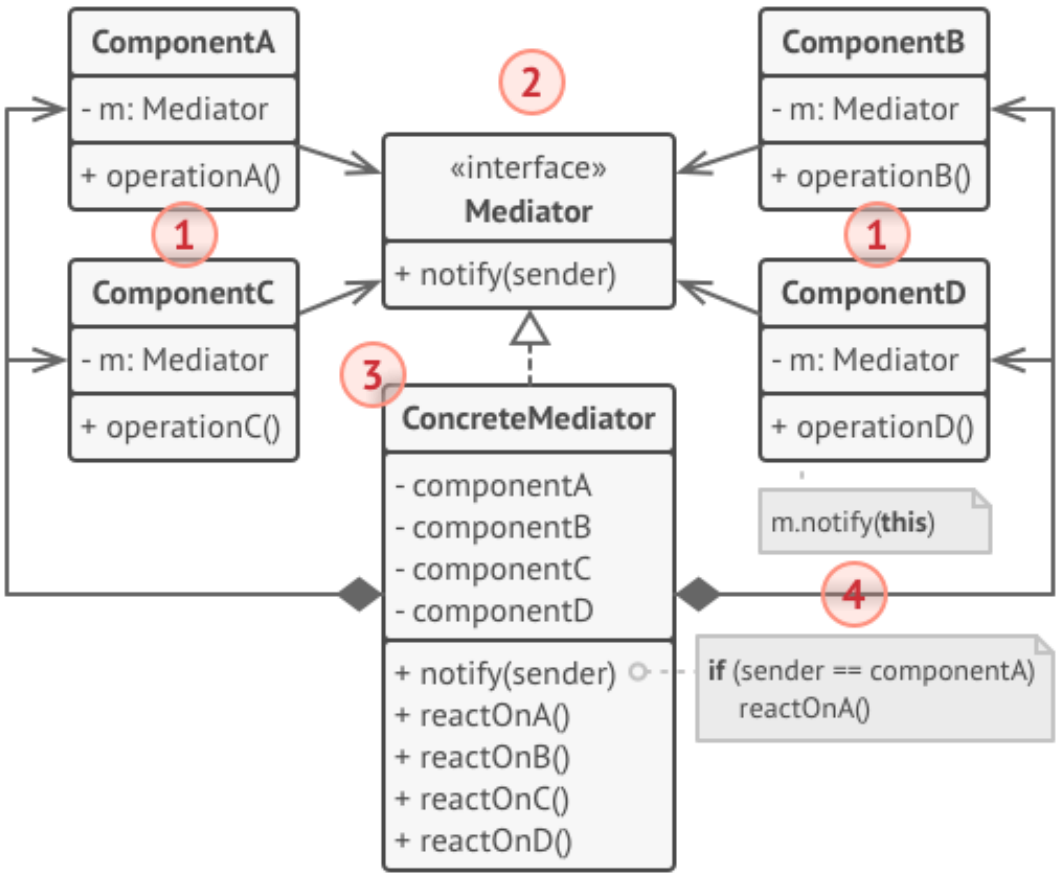

Рисунок 29 – Структура паттерна «посредник»

– компоненты — это разнородные объекты, содержащие бизнес-логику программы. У каждого компонента есть ссылка на объект-посредник;
– интерфейс паттерна, использующийся для обмена информации между различными компонентами системы. Как правило, содержит один метод уведомления о различных событиях. В этот метод передаются детали определённого события: ссылку на компонент-уведомитель, и любые другие данные;
– реализация интерфейса посредника;
– все компоненты общаются через класс посредника, а не напрямую. Когда происходит важное событие, компонент оповещает своего посредника, а посредник сам решает кому и что передавать.

В программном коде реализация паттерна показана на рисунке 30.

```
public interface Mediator {
    void addNewImage(Image image);

    void deleteNote();

    void getInfoFromList(Image image);

    void saveChanges();

    void markNote();

    void clear();

    void setElementsList(ListModel<?> list);

    void registerComponent(Component component);

    void hideElements(boolean flag);

    void createGUI();
}
```

Рисунок 30 – Код интерфейса посредника

Как видно из рисунка 30, некоторые методы принимают на вход объект типа Image, который отвечает за свойства загружаемых в программу изображений [19]. На рисунке 31 представлен код класса Image, который имеет 3 поля (название и путь до файла и сам файл изображения) и 7 методов для работы с ним. Все эти методы являются геттерами и сеттерами полей класса.

```
public class Image {
    private String name;
    private String path;
    private File file;

    public Image() {
        name =
        path =
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setPath(String path) {
        this.path = path;
    }

    public void setFile(File file) {
        this.file = file;
    }

    public String getName() {
        return name;
    }

    public String getPath() {
        return path;
    }

    public File getFile() {
        return file;
    }

    @Override
    public String toString() {
        return name;
    }
}
```

Рисунок 31 – Программный код класса Image

Реализация интерфейса посредника происходит в классе Editor, который содержит функционал регистрирования конкретных компонентов программы. Регистрация компонентов происходит в точке входа в программу

– классе Main. На рисунке 32 представлен программный код метода registerComponent.

```
@Override
public void registerComponent(Component component) {
    component.setMediator(this);
    switch (component.getName()) {
        case "AddButton":
            add = (AddButton) component;
            break;
        case "OrbButton":
            orb = (OrbButton) component;
            break;
        case "SurfButton":
            surf = (SurfButton) component;
            break;
        case "SiftButton":
            sift = (SiftButton) component;
            break;
        case "BriefButton":
            brief = (BriefButton) component;
            break;
        case "ImagePanel":
            imagePanel = (ImagePanel) component;
            break;
        case "DeleteButton":
            del = (DeleteButton) component;
            break;
        case "List":
            list = (List) component;
            this.list.addListSelectionListener(listSelectionEvent -> {
                Image image = (Image) list.getSelectedValue();
                if (image != null) {
                    getInfoFromList(image);
                } else {
                    clear();
                }
            });
            break;
        case "TextBox":
            textBox = (TextBox) component;
            break;
        case "Title":
            title = (Title) component;
            break;
    }
```

Рисунок 32 – Программный код метода registerComponent

Таким образом, реализация паттерна «посредник» позволила упростить читаемость и связность кода в приложении.

Следующим шагом является реализация самих дескрипторов особых точек. На рисунке 33 представлен программный код дескриптора особых точек BRIEF.

```
public class Brief {
    /**
     * Реализация дескриптора BRIEF
     *
     * @param input входное изображение
     * @return обработанное изображение
     */
    public Mat run(Mat input) {
        long time = System.currentTimeMillis();
        int hessianThreshold = 400;
        int nOctaves = 4, nOctaveLayers = 3;
        MatOfKeyPoint keypoints = new MatOfKeyPoint();
        Mat descriptor = new Mat();
        Mat output = new Mat();
        BRISK brisk = BRISK.create(hessianThreshold, nOctaves, nOctaveLayers);
        brisk.detectAndCompute(input, new Mat(), keypoints, descriptor);
        Features2d.drawKeypoints(input, keypoints, output);
        time = System.currentTimeMillis() - time;
        System.out.println("Processing time in ms = " + time);
        System.out.println("Number of keypoints = " + keypoints.toList().toArray().length);
        return output;
    }
```

Рисунок 33 – Программный код дескриптора BRIEF

Как видно из рисунка 33, данный код задает начальные параметры для работы дескриптора, которые были описаны в математической модели ранее: количество октав и их слоёв, а также порог фильтрации.

Остальные дескрипторы имеют аналогичную реализацию и имеют похожие параметры инициализации: 4 октавы и 3 слоя октав, с порогом фильтрации в 400 единиц.

Рассмотрим элементы графического интерфейса программы. На рисунке 34 представлен программный код кнопки «добавить».

```
public class AddButton extends JButton implements Component {
    private Mediator mediator;

    public AddButton() {
        super("Add");
    }

    @Override
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    @Override
    protected void fireActionPerformed(ActionEvent actionEvent) {
        JFileChooser fileChooser = new JFileChooser(FileSystemView.getFileSystemView());
        int ret1 = fileChooser.showDialog(null, "Открыть файл");
        if (ret1 == JFileChooser.APPROVE_OPTION) {
            File file1 = fileChooser.getSelectedFile();
            Image img = new Image();
            img.setName(file1.getName());
            img.setPath(file1.getPath());
            img.setFile(file1);
            mediator.addNewImage(img);
        }
    }

    @Override
    public String getName() {
        return "AddButton";
    }
}
```

Рисунок 34 – Программный код кнопки «добавить»

Все элементы интерфейса реализуют функционал объектов библиотеки Swing. В данном случае класс наследуется от класса JButton. Кнопка работает с объектами типа Image. Который был описан ранее. Также необходимо создать связь с классом-посредником Mediator, являющимся связующим звеном всех элементов графического интерфейса.

На рисунке 35 представлен программный код кнопки добавить.

```
private Mediator mediator;

public BriefButton() {
    super("Brief");
}

@Override
public void setMediator(Mediator mediator) {
    this.mediator = mediator;
}

@Override
protected void fireActionPerformed(ActionEvent actionEvent) {
    Brief brief = new Brief();
    Mat processedImage = brief.run(Imgcodecs.imread(IMAGE_PATH));
    ImageHelper imageHelper = new ImageHelper();
    imageHelper.addImage(processedImage);
}

@Override
public String getName() {
    return "BriefButton";
}
```

Рисунок 35 – Программный код кнопки «добавить»

Также необходимо упомянуть класс Component, задающий функции, которые необходимо реализовать в каждом объекте графического интерфейса. Метод GetName() должен возвращать имя компонента в строковом представлении, а метод setMediator(Mediator mediator) должен инициализировать объект посредника для дальнейшей работы. Код представлен на рисунке 36.

```
package components;

import mediator.Mediator;

public interface Component {
    void setMediator(Mediator mediator);

    String getName();
}
```

Рисунок 36 – Программный код интерфейса Component

Далее рассматривается программный код кнопки «удалить». Как и большинство других классов, он наследуется от JButton и реализует функционал интерфейса Component. Код представлен на рисунке 37.

```
public class DeleteButton extends JButton implements Component {
    private Mediator mediator;

    public DeleteButton() {
        super("Delete");
    }

    @Override
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    @Override
    protected void fireActionPerformed(ActionEvent actionEvent) {
        mediator.deleteNote();
    }

    @Override
    public String getName() {
        return "DeleteButton";
```

Рисунок 37 – Программный код кнопки удаления

Также на рисунке 38 представлен программный код поля Title.

```
public class Title extends JTextField implements Component {
    private Mediator mediator;

    @Override
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    @Override
    protected void processComponentKeyEvent(KeyEvent keyEvent) {
        mediator.markNote();
    }

    @Override
    public String getName() {
        return "Title";
```

Рисунок 38 – Программный код поля Title

Класс Title является наследником класса JTextField, являющимся одним из компонентов библиотеки Swing. Этот класс отвечает за создание пустого текстового поля со строкой.

Класс ImageHelper отвечает за создание основного окна программы. Он является предком класса JFrame, являющегося реализацией главного окна программы.

```
public class ImageHelper extends JFrame implements Component {

    private Mediator mediator;
    private final JFrame frame = new JFrame();
    private final JPanel panel = new JPanel();
    private final JScrollPane scrollPane = new JScrollPane(panel);

    public ImageHelper() {
        panel.setLayout(new FlowLayout());
        frame.setSize(640, 450);
        frame.setDefaultCloseOperation(WindowConstants.HIDE_ON_CLOSE);
        frame.add(scrollPane);
    }

    /**
     * добавление входного изображение и отображение окна
     *
     * @param inputImage входное изображение
     */
    public void addImage(Mat inputImage) {

        int type = BufferedImage.TYPE_BYTE_GRAY;

        if (inputImage.channels() > 1) {
            type = BufferedImage.TYPE_3BYTE_BGR;
        }
        int bufferSize = inputImage.channels() * inputImage.cols() * inputImage.rows();
        byte[] b = new byte[bufferSize];

        inputImage.get(0, 0, b);
        BufferedImage image = new BufferedImage(inputImage.cols(), inputImage.rows(), type);
        final byte[] targetPixels = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
        System.arraycopy(b, 0, targetPixels, 0, b.length);

        ImageIcon imageIcon = new ImageIcon(image);

        JLabel label = new JLabel();
        label.setIcon(imageIcon);

        panel.add(label);
        frame.setVisible(true);
    }
```

Рисунок 39 – Программный код класса ImageHelper

Как видно из рисунка 39, в начале класса объявляются и инициализируются объекты типа JFrame, JPanel и JScrollPane. Объект frame отвечает за параметры основного окна программы, в то время как объект panel создаёт панель на главном окне, а объект scrollPane – полосу прокрутки для панели. Далее идёт подготовка панели к выводу изображения в интерфейс.

Класс ImagePanel выполняет роль панели, которая создается в классе ImageHelper.

```
public class ImagePanel extends JPanel implements Component {

    private Mediator mediator;
    private BufferedImage image;

    public ImagePanel() {
        try {
            image = ImageIO.read(new File());
        } catch (IOException ex) {
            System.out.println("Error during reading image.");
        }
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(image, 0, 0, this);
    }

    @Override
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    @Override
    public String getName() {
        return "ImagePanel";
    }
```

Рисунок 40 – Программный код класса ImagePanel

Рисунок 40 показывает, что данный класс выполняет метод paintComponent, отвечающий за создание изображения в конкретном месте панели.

Следующим в очереди необходимо рассмотреть класс List, реализующий список изображений.

```
public class List extends JList<Object> implements Component {
    private Mediator mediator;
    private final DefaultListModel<Object> LIST_MODEL;

    public List(DefaultListModel listModel) {
        super(listModel);
        this.LIST_MODEL = listModel;
        setModel(listModel);
        this.setLayoutOrientation(JList.VERTICAL);
        Thread thread = new Thread(new Hide(this));
        thread.start();
    }

    @Override
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    public void addElement(Image image) {
        LIST_MODEL.addElement(image);
        int index = LIST_MODEL.size() - 1;
        setSelectedIndex(index);
        ensureIndexIsVisible(index);
    }

    public void deleteElement() {
        int index = this.getSelectedIndex();
        try {
            LIST_MODEL.remove(index);
        } catch (ArrayIndexOutOfBoundsException ignored) {
        }
    }

    public Image getCurrentElement() {
        return (Image) getSelectedValue();
    }
```

Рисунок 41 – Публичные методы класса List

На рисунке 41 представлены публичные методы класса List. Этот класс наследует класс JPanel библиотеки Swing и реализует методы интерфейса Component. В конструкторе класса задаётся модель и её ориентация, а также объект Thread, отвечающий за работу с потоками. Метод addElement (Image image) реализует логику добавления изображения в список. Метод deleteElement() выполняет логику удаления изображения из списка путем работы с его индексом. Метод getCurrentElement() возвращает текущее изображение в виде объекта типа Image.

Далее рассмотрим программный код класса Hide, представленный на рисунке 42.

```
private class Hide implements Runnable {
    private List list;

    Hide(List list) {
        this.list = list;
    }

    @Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(300);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
            mediator.hideElements(list.isSelectionEmpty());
        }
    }
}
```

Рисунок 42 – Программный код класса Hide

Этот класс является приватным и взаимодействует только с классом List. Единственная функция класса Hide – скрытие элементов из списка. Это реализовано путем управления потоком, который каждые 300 миллисекунд проверяет элементы списка.

Следующим в очереди является класс SaveButton, отвечающий за функционал кнопки сохранения. Программный код представлен на рисунке 43.

```
public class SaveButton extends JButton implements Component {
    private Mediator mediator;

    public SaveButton() {
        super("Save");
    }

    @Override
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    @Override
    protected void fireActionPerformed(ActionEvent actionEvent) {
        JFileChooser fileChooser = new JFileChooser(FileSystemView.getFileSystemView());
        int ret1 = fileChooser.showDialog(null, "Сохранить файл");
        if (ret1 == JFileChooser.APPROVE_OPTION) {
            File file1 = fileChooser.getSelectedFile();
        }
    }

    @Override
    public String getName() {
        return "SaveButton";
    }
}
```

Рисунок 43 – Программный код класса SaveButton

Класс является потомком класса JButton и реализует весь интерфейс Component. Метод fireActionPerformed (ActionEvent actionEvent) реагирует на событие нажатия кнопки и при совершении этого действия открывает файловый проводник для выбора места сохранения изображения.

Одним из последних элементов графического интерфейса программы является класс TextBox. Программный код этого класса представлен на рисунке 44.

```
public class TextBox extends JTextArea implements Component {
    private Mediator mediator;

    @Override
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    @Override
    protected void processComponentKeyEvent(KeyEvent keyEvent) {
        mediator.markNote();
    }

    @Override
    public String getName() {
        return "TextBox";
    }
}
```

Рисунок 44 – Программный код класса TextBox

Класс наследуется от JTxtArea, являющегося многострочным текстовым полем.

На рисунке 45 представлен код кнопки SIFT.

```
public SiftButton() {
    super("Sift");
}

@Override
public void setMediator(Mediator mediator) {
    this.mediator = mediator;
}

@Override
protected void fireActionPerformed(ActionEvent actionEvent) {
    Sift sift = new Sift();
    Mat processedImage = sift.run(Imgcodecs.imread(IMAGE_PATH));
    ImageHelper imageHelper = new ImageHelper();
    imageHelper.addImage(processedImage);
}

@Override
public String getName() {
    return "SiftButton";
}
```

Рисунок 45 – Программный код класса SiftButton

Как и остальные кнопки дескрипторов, эта кнопка выполняет код функции fireActionPerformed() при нажатии на неё. Это последний рассматриваемый элемент графического интерфейса. Кнопка взаимодействует с классом Sift, который рассматривается далее.

Класс Sift отвечает за логику и входные параметры дескриптора особых точек Sift. Как и остальные дескрипторы, на вход подается 3 слоя по 4 октавы в каждом. Метод run рассчитывает время выполнения программы и количество найденных особых точек. Программный код класса Sift представлен на рисунке 46.

```
public class Sift {
    /**
     * Реализация дескриптора SIFT
     *
     * @param input входное изображение
     * @return обработанное изображение
     */
    public Mat run(Mat input) {
        long time = System.currentTimeMillis();
        int hessianThreshold = 400;
        int nOctaves = 4, nOctaveLayers = 3;
        MatOfKeyPoint keypoints = new MatOfKeyPoint();
        Mat descriptor = new Mat();
        Mat output = new Mat();
        SIFT sift = SIFT.create(hessianThreshold, nOctaves, nOctaveLayers);
        sift.detectAndCompute(input, new Mat(), keypoints, descriptor);
        Features2d.drawKeypoints(input, keypoints, output);
        time = System.currentTimeMillis() - time;
        System.out.println("Processing time in ms = " + time);
        System.out.println("Number of keypoints = " + keypoints.toList().toArray().length);
        return output;
    }
}
```

Рисунок 46 – Программный код класса Sift

Последним элементом рассматриваемого кода является точка входа программы. В методе Main в первую очередь загружается библиотека OpenCV через метод System.loadLibrary. Далее инициализируется объект-посредник и

регистрируются все компоненты интерфейса. В конце концов, метод createGUI() создает графический интерфейс программы.

```
import components.*;
import mediator.*;
import org.opencv.core.Core;

import javax.swing.*;

public class Main {

    public static void main(String[] args) {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
        Mediator mediator = new Editor();
        mediator.registerComponent(new Title());
        mediator.registerComponent(new TextBox());
        mediator.registerComponent(new AddButton());
        mediator.registerComponent(new OrbButton());
        mediator.registerComponent(new SurfButton());
        mediator.registerComponent(new BriefButton());
        mediator.registerComponent(new SiftButton());
        mediator.registerComponent(new DeleteButton());
        mediator.registerComponent(new List(new DefaultListModel<>()));
        mediator.registerComponent(new ImagePanel());

        mediator.createGUI();
    }
}
```

Рисунок 47 – Программный код точки входа программы

Таким образом, был рассмотрен весь код разработанного программного обеспечения.

3.3 Проведение сравнительного анализа дескрипторов

На основе разработанного программного обеспечения фиксируются результаты работы различных дескрипторов особых точек на изображениях с лицами.

Для начала рассмотрим работу программы. На рисунке 48 представлено основное окно программы до выбора обрабатываемой картинки. В левой части присутствуют кнопки добавления и удаления картинок из списка.

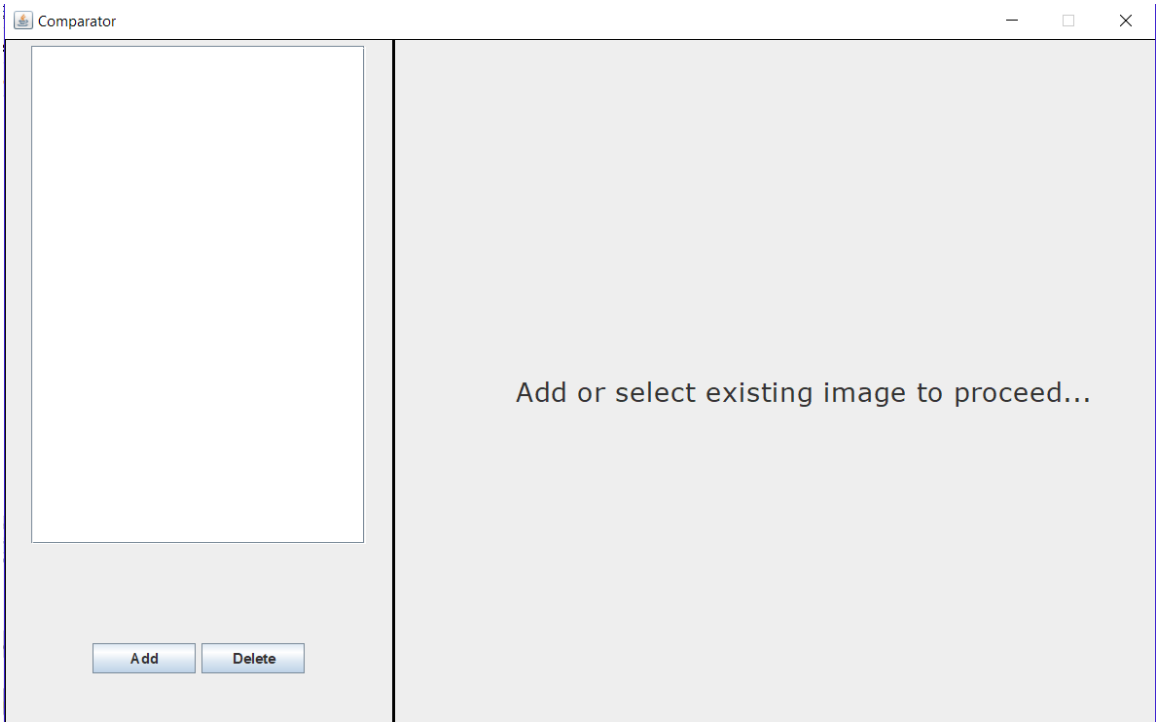

Рисунок 48 – Основное окно программы до выбора изображения

Далее необходимо нажать кнопку add и выбрать нужное изображение в файловой системе. После откроется доступ к правой части программы, которая содержит превью выбранного из списка изображения и доступные методы обработки изображения. Пример окна программы после выбора изображения представлен на рисунке 49.

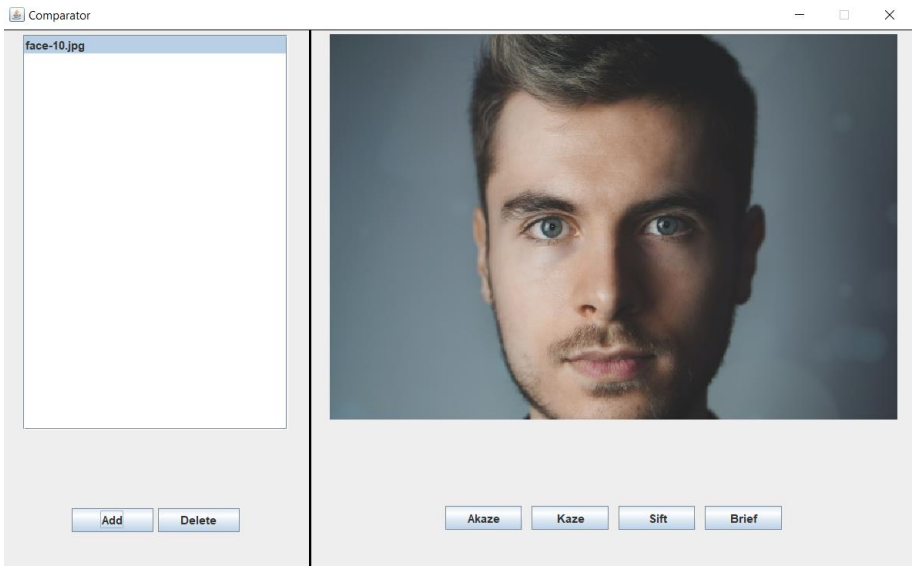

Рисунок 49 – Основное окно программы после выбора изображения

После выбора изображения его нужно обработать одним из дескрипторов. На рисунке 50 представлено изображение, обработанное дескриптором особых точек. Дескриптор отображает все найденные особые точки на изображении.

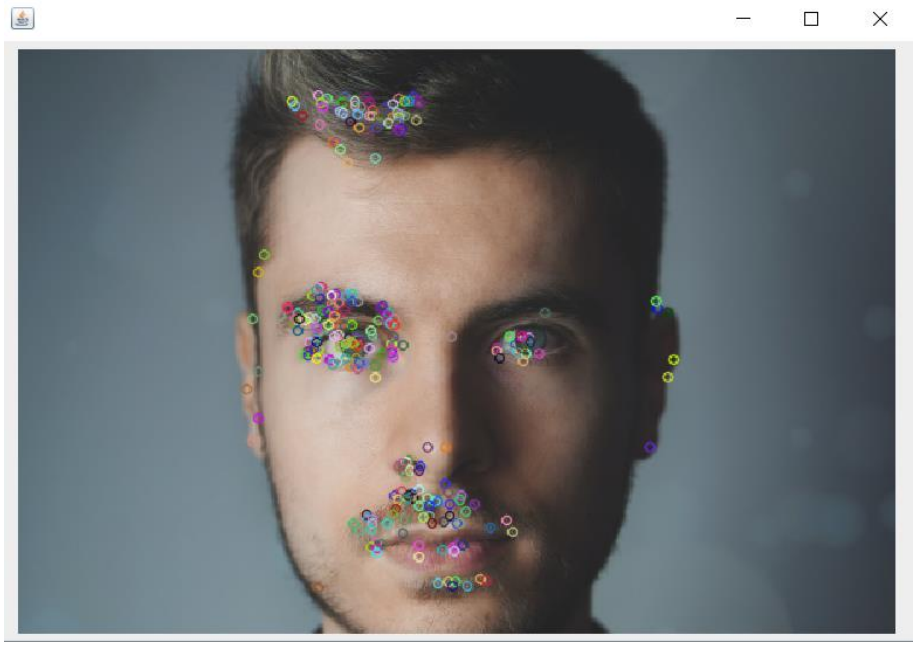

Рисунок 50 – Обработанное изображение

В качестве экспериментальной выборки сгенерированы 3 различных набора изображений. На каждом изображении из набора зафиксированы результаты и на их основе составлены средние результаты по скорости и точности работы дескрипторов на конкретном наборе.

Первым набором изображений является yalefaces. Это один из самых старых и надёжных наборов, сделанный в 2003 году. Он состоит из лиц, снятых при разном освещении и с разными окклюзиями, такими как очки. Набор изображений представлен на рисунке 51.

Рисунок 51 – Набор изображений yalefaces

Следующим набором изображений является flickr faces. Набор содержит 70000 изображений в разрешении 1024 на 1024 пикселей и содержит большое количество лиц с окклюзиями, например, очки, шляпы, повязки и т.д. В нём содержатся изображения людей разных национальностей, типажей и возрастов, и это сильно повышает универсальность обученной модели. В данной работе обрабатывается незначительная часть исходного набора, так как обработать весь набор данных не представляется возможным. Набор изображений представлен на рисунке 52.

Рисунок 52 – Набор изображений flickr faces

Третьим набором изображений является tufts database [30]. Особенностью данного набора является применение нестандартных подходов к получению изображений. Лица участвующих обрабатывались с помощью специальных фильтров для усложнения распознавания. Также участники снимали через тепловизор и различные цветовые фильтры. На рисунке 53 представлен набор изображений tufts faces.

Рисунок 53 – Набор изображений tufts faces

После выбора и получения соответствующих наборов изображений, необходимо провести расчёты с использованием ранее реализованных дескрипторов особых точек.

Для начала необходимо узнать чувствительность (recall) каждого из дескрипторов. Для этого общее количество найденных особых точек делится на заранее известное количество правильных особых точек на изображении. Чувствительность помогает понять, насколько дескриптор устойчив к окклюзиям на изображении. Результаты представлены в таблице 4.

Таблица 4 – Средняя чувствительность дескрипторов

Дескриптор / Датасет	BRIEF	SIFT	AKAZE
yalefaces	0,96	1,06	0,99
flickr faces	0,88	0,92	0,96
tufts faces	0,94	0,95	1,05

Как видно из таблицы 4, наиболее чувствительным дескриптором на наборе изображений yalefaces AKAZE с отклонением всего в 0,01 от нормы. На наборе изображений flickr faces наилучший результат также показал AKAZE, в то время как BRIEF обладает отклонением в 0,12 пунктов. На наборе изображений tufts faces одинаково хорошо показали себя дескрипторы SIFT и AKAZE с отклонением в 0,05. Набор изображений yalefaces является наиболее легким для корректного распознавания особых точек ввиду своей простоты. На изображениях нету серьёзных окклюзий и фильтров.

Также необходимо отметить, что количество найденных особых точек и, соответственно, чувствительность дескрипторов существенно падает при наличии окклюзий на изображении (очки, блики, шляпы) на всех наборах изображений. Однако ввиду большой выборки, эти аномалии не оказывают сильное влияние на финальные результаты.

На основе вышеописанной таблицы можно построить гистограмму чувствительности дескрипторов. Результаты будут сгруппированы по каждому из дескрипторов. Гистограмма представлена на рисунке 54.

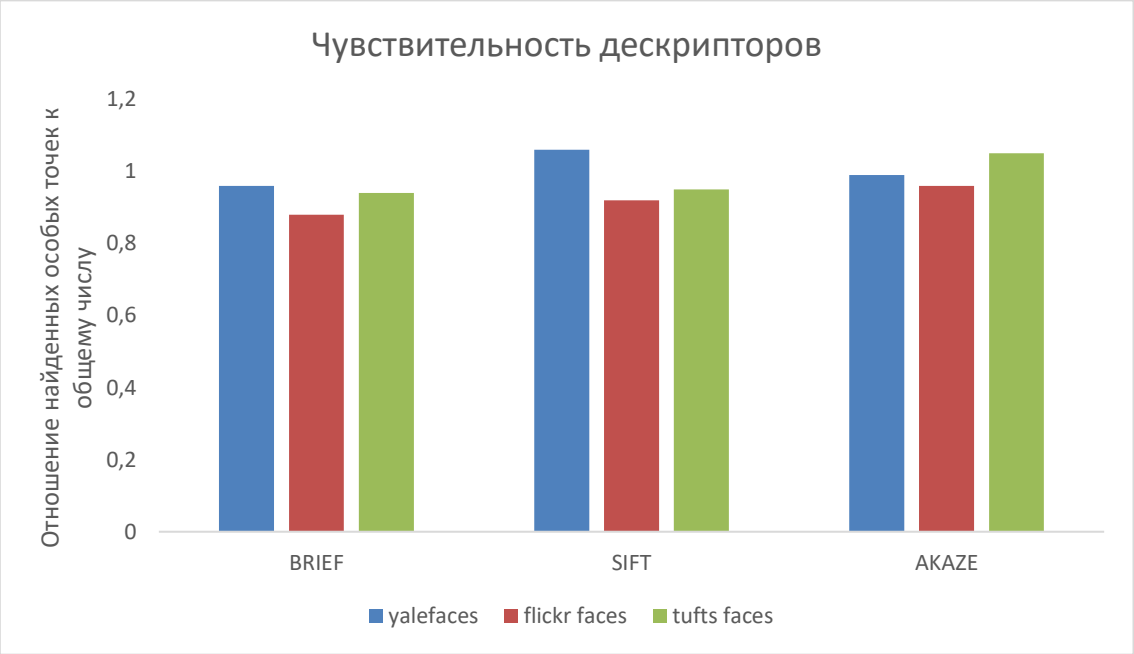

Рисунок 54 – Гистограмма чувствительности дескрипторов

Как видно из рисунка 54, наиболее чувствительным из дескрипторов является AKAZE, а наименее чувствительным – BRIEF.

Теперь рассмотрим среднюю скорость работы дескрипторов. Для каждого изображения в наборе замеряется время его обработки, а потом рассчитывается среднее значение на всём наборе. Результаты представлены в таблице 5.

Таблица 5 – Средняя время обработки изображения, мс

Дескриптор / Датасет	BRIEF	SIFT	AKAZE
yalefaces	121	145	109
flickr faces	256	252	198
tufts faces	370	341	288

Как видно из таблицы 5, наиболее быстрым дескриптором на всех наборах данных является AKAZE. Самым медленным дескриптором при обработке набора изображений yalefaces явялется SIFT, а на наборе изображений flickr faces хуже остальных показал себя BRIEF. Набор изображений tufts faces имеет самое высокое среднее время обработки среди всех дескрипторов. Хуже всего с ним справился BRIEF.

На основе таблицы 5 можно построить гистограмму быстродействия дескрипторов. Данные в гистограмме сгруппированы по дескрипторам, позволяя сделать общие выводе о скорости работы каждого из них. Гистограмма представлена на рисунке 55.

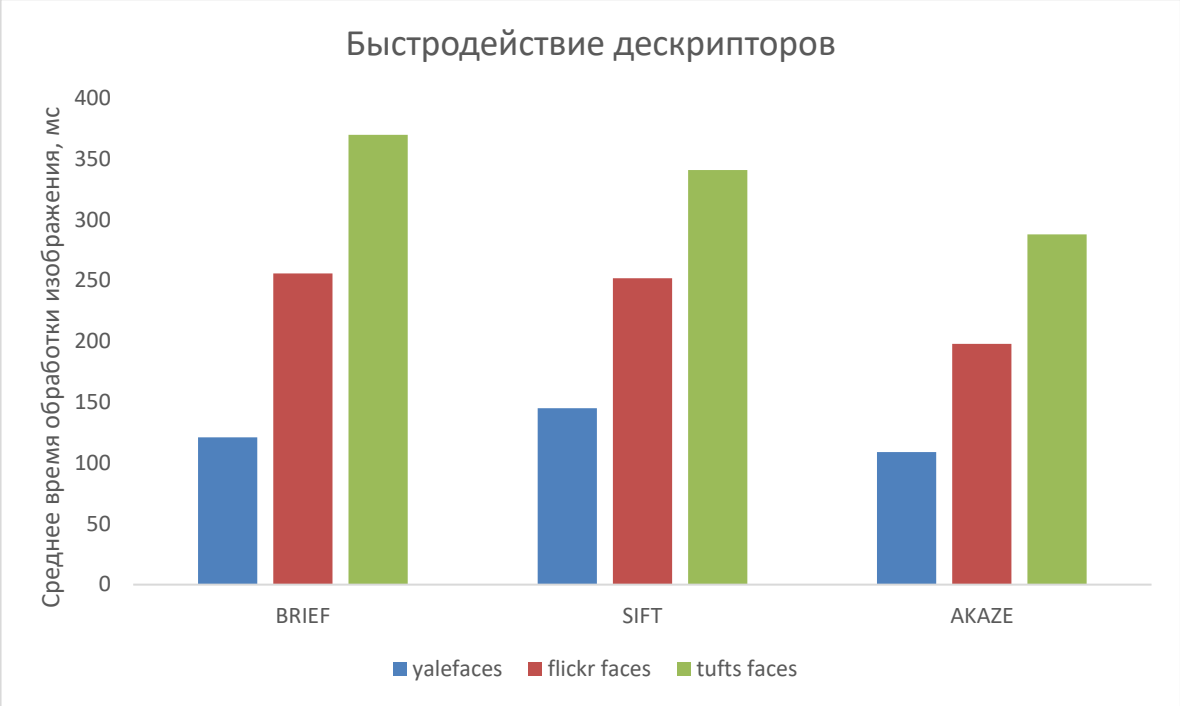

Рисунок 55 – Гистограмма быстродействия дескрипторов

Как видно из рисунка 55, дескриптор AKAZE имеет самое низкое время обработки на всех наборах изображений и соответственно является самым быстрым. Самым медленным дескриптором оказался BRIEF, который имеет самое высокое время обработки на всех наборах изображений, кроме yalefaces.

Последним анализируемым параметром является точность (precision) дескрипторов. Результаты точности представлены в таблице 6.

Таблица 6 – Точность дескрипторов

Дескриптор / Датасет	BRIEF	SIFT	AKAZE
yalefaces	0,958	0,943	0,99
flickr faces	0,864	0,913	0,958
tufts faces	0,936	0,947	0,952

Точность дескрипторов находится путём деления неправильно найденных особых точек на все найденные особые точки. Полученный результат вычитается из единицы. Таким образом, получается значение от 0 до 1, которое отображает точность дескриптора. Как видно из таблицы 6, наилучшую точность на всех наборах изображений имеет дескриптор AKAZE. Самую низкую возможную точность показал дескриптор BRIEF на наборе изображений flickr faces, равную 0,864. На наборе изображений yalefaces хуже остальных показал себя дескриптор SIFT.

На основе таблицы 6 построена гистограмма точности дескрипторов. В ней результаты сгруппированы по дескрипторам особых точек. Гистограмма представлена на рисунке 56.

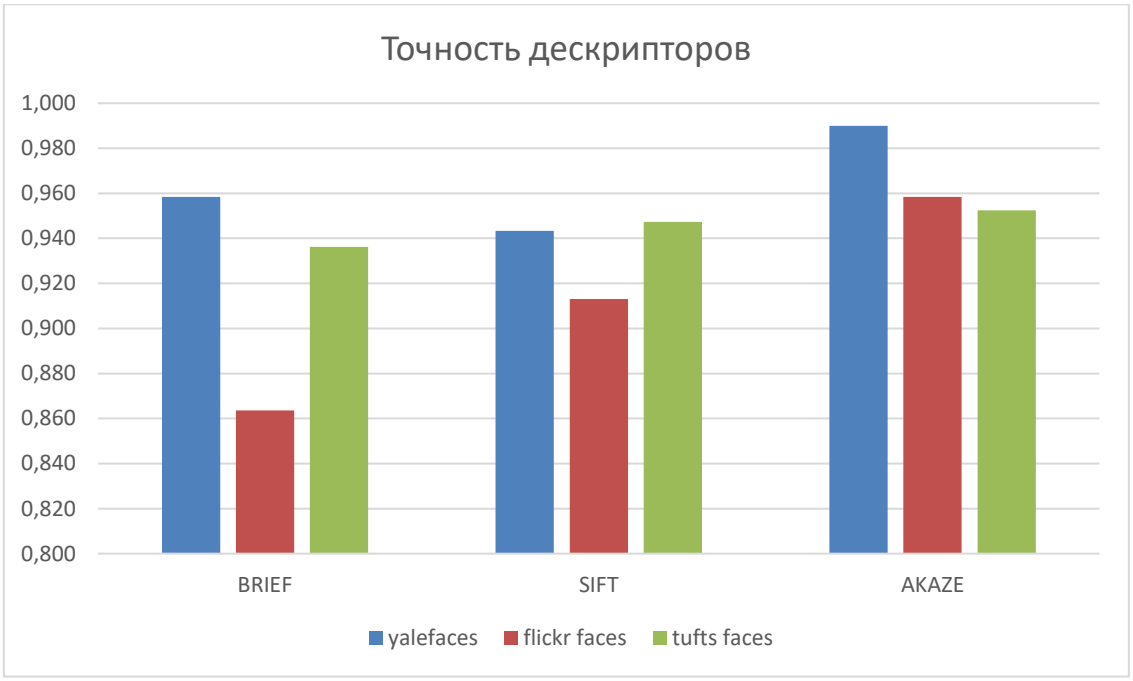

Рисунок 56 – Гистограмма точности дескрипторов

По гистограмме можно понять, что самым точным дескриптором особых точек для распознавания лиц является AKAZE, а наименее точным – BRIEF.

Из наблюдений можно сделать следующие выводы:

– наиболее чувствительным дескриптором является AKAZE. Это означает, что он лучше остальных дескрипторов приспособлен для распознавания лиц с окклюзиями (очки, шляпы, освещение);
– чувствительность и точность всех дескрипторов существенно падает при наличии окклюзий на изображении;
– самым быстрым дескриптором особых точек является AKAZE, а самым медленным – BRIEF;
– дольше всего обрабатываются изображения из набора данных tufts faces, а быстрее всего – изображения из набора yalefaces;
– самым точным дескриптором особых точек для распознавания лиц является AKAZE, а наименее точным – BRIEF;
– наибольшая точность распознавания достигается на наборе изображений yalefaces, а наименьшая – на наборе flickr faces.

Заключение

Данная магистерская работа посвящена исследованию эффективности дескрипторов особых точек. Задача распознавания лиц актуальная во многих сферах нашей жизни. Однако дескрипторы особых точек, использующиеся для этих задач, имеют свои преимущества и недостатки. Отсутствие точной информации о нюансах работы каждого из дескрипторов значительно снижают эффективность и точность систем распознавания лиц. Благодаря данной работе получены новые результаты работы дескрипторов особых точек, а также разработано программное обеспечение для быстрого и эффективного сравнения.

В первом разделе рассматривается предметная область исследования и проводится постановка задачи.

Во втором разделе раскрываются основные этапы работы дескрипторов особых точек, а также формулируется математическая модель для каждого из дескрипторов и всей системы в целом.

В третьем разделе описывается программная реализация математической модели и фиксируются результаты работы различных дескрипторов особых точек. На основе этих результатов можно сделать следующие выводы:

- дескриптор AKAZE является самый быстрым и точным дескриптором на большинстве наборах изображений;
- дескриптор BRIEF является наименее точным и наиболее медленным среди всех рассматриваемых дескрипторов;
- чувствительность и точность всех дескрипторов существенно падает при наличии окклюзий на изображении;
- на конкретных наборах изображений точность дескриптора SIFT значительно падает по сравнению с другими дескрипторами.

Список используемой литературы

1. Визильтер Ю. В. Обработка и анализ изображений в задачах машинного зрения: Курс лекций и практических занятий. – М.: Физматкнига, 2010. – 672 с.
2. Гонсалес Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. – Москва: Техносфера, 2012. – 1104 с.
3. И. С. Грузман Цифровая обработка изображений в информационных системах / Грузман И.С., Киричук В.С., Косых В.П., Перетягин Г.И., Спектор А.А.: Учебное пособие - Новосибисрк: Изд-во НГТУ, 2002. - 352 с.
4. Использование GPU для решения задач компьютерного зрения в библиотеке OpenCV [Электронный ресурс] / Режим доступа: http://agora.guru.ru/hpc-h/files/Using_GPU_for_computer_vision_in_OpenCV_library.pdf
5. Клейнберг Дж., Тардос Е. Алгоритмы: разработка и применение. Классика Computers Science / Пер. с англ. Е. Матвеева. – СПб.: Питер, 2016. – 800 с.
6. Краснобаев, Е.А. Сравнение бинарных дескрипторов особых точек изображений в условиях искажений / Е.А. Краснобаев, Д.В. Чистобаев, А.Л. Малышев // Компьютерная оптика. – 2019. – Т. 43, № 3. – С. 434-445.
7. Машинное зрение: понятия, задачи и области применения [Электронный ресурс] / Режим доступа: http://rusnauka.com/25_SSN_2009/Informatica/51050.doc.htm.
8. Н. Красильников Цифровая обработка 2D- и 3D-изображений / Красильников Н.: Отдельное издание. - БХВ-Петербург, 2016. - 608 с.
9. Паттерн посредник [Электронный ресурс] / Режим доступа: https://refactoring.guru/ru/design-patterns/mediator

10. Системы компьютерного зрения: современные задачи и методы [Электронный ресурс] / Режим доступа: http://controleng.ru/innovatsii/sistemy-komp-yuternogo-zreniya-sovremenny-e-zadachi-metody/.

11. Что такое машинное зрение и чем оно отличается от человеческого? [Электронный ресурс] / Режим доступа: https://meduza.io/feature/2019/03/30/chto-takoe-mashinnoe-zrenie-i-chem-ono-otlichaetsya-ot-chelovecheskogo-seychas-ob-yasnim-ponyatno

12. A tutorial on binary descriptors [Электронный ресурс] / Режим доступа: https://gilscvblog.com/2013/10/04/a-tutorial-on-binary-descriptors-part-3-the-orb-descriptor/

13. Beyerer J. Automated Visual Inspection: Theory, Practice and Applications / J. Beyerer, P. Fernando, F. Christian. – Springer Berlin Heidelberg, 2016. – 798 p.

14. Calonder M. BRIEF: binary robust independent elementary features / M. Calonder, V. Lepetit, C. Strecha, P. Fua // European Conference on Computer Vision, – 2010. – P. 778-792.

15. Face Recognition Based on Texture Descriptors [Электронный ресурс] / Режим доступа: https://www.intechopen.com/chapters/60862

16. Faster and better: a machine learning approach to corner detection [Электронный ресурс] / Режим доступа: https://arxiv.org/pdf/0810.2434.pdf

17. Feature detection and description [Электронный ресурс] / Режим доступа: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html

18. Fusing Points and Lines for High Performance Tracking [Электронный ресурс] / Режим доступа: http://www.edwardrosten.com/work/rosten_2005_tracking.pdf

19. James L. Pro JavaFX 2/ L. James, G. Weiqi, C. Stephen, I. Dean, V. Johan. – Apress, 2012. – 641 p.

20. Just look at the image: viewpoint-specific surface normal prediction for improved multi-view reconstruction [Электронный ресурс] / Режим доступа: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Galliani_Just_Look_at_CVPR_2016_paper.pdf

21. Introduction To Feature Detection And Matching [Электронный ресурс] / Режим доступа: https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d

22. Introduction to SURF [Электронный ресурс] / Режим доступа: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

23. Machine learning for high-speed corner detection [Электронный ресурс] / Режим доступа: http://www.edwardrosten.com/work/rosten_2006_machine.pdf

24. Mediator Design Pattern [Электронный ресурс] / Режим доступа: https://sourcemaking.com/design_patterns/mediator

25. ORB: Oriented FAST and Rotated BRIEF [Электронный ресурс] / Режим доступа: http://www.willowgarage.com/sites/default/files/orb_final.pdf

26. S. M. M. Kahaki, Contour-Based Corner Detection and Classification by Using Mean Projection Transform, 2014.

27. SIFT: Theory and practice [Электронный ресурс] / Режим доступа: https://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/

28. SURF: Speeded Up Robust Features [Электронный ресурс] / Режим доступа: http://people.ee.ethz.ch/~surf/eccv06.pdf

29. TIOBE index [Электронный ресурс] / Режим доступа: https://www.tiobe.com/tiobe-index/

30. Tufts database [Электронный ресурс] / Режим доступа: http://tdface.ece.tufts.edu/