Применение функций OpenCV в компьютерном зрении (60 примеров на Python)

Book · January 2022

CITATIONS
0

READS
899

1 author:

Sergey Aleksandrovih Molodyakov
Peter the Great St.Petersburg Polytechnic University
58 PUBLICATIONS **96** CITATIONS

SEE PROFILE

САНКТ–ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

С.А. МОЛОДЯКОВ

ПРИМЕНЕНИЕ ФУНКЦИЙ OPENCV В КОМПЬЮТЕРНОМ ЗРЕНИИ
(60 примеров на Python)

Санкт-Петербург
Издательство Политехнического университета
2022

УДК 004.932

Рецензенты:
С.М. Устинов - д.т.н., профессор высшей школы программной инженерии СПбПУ
А.П. Лавров - д.т.н., профессор высшей школы прикладной физики и космических технологий СПбПУ

Применение функций OpenCV в компьютерном зрении (60 примеров на Python) / Молодяков С.А. СПб.: Изд-во Политехн. ун-та, 2022.- 296 с.

В монографии излагаются вопросы обработки изображений с применением функций библиотеки OpenCV. Представлены 60 примеров программ, написанных на языке Python. Примеры разделены на три уровня алгоритмов: простой низкоуровневой обработки изображений, сегментирования и высокоуровневой обработки (распознавания); а также обработки видео. Представленные примеры являются рабочими программами, которые написаны студентами высшей школы программной инженерии Санкт-Петербургского политехнического университета.

Монография предназначена для начинающих инженеров и специалистов в области проектирования и применения видеосистем для обработки изображений. Примеры программ позволяют упростить изучение алгоритмов компьютерного зрения и функций библиотеки OpenCV, а также процесс экспериментирования над изображениями. Монография может быть полезна студентам, проходящим подготовку по направлениям 09.03.01 «Информатика и вычислительная техника», 09.03.04 «Программная инженерия»,12.03.01 «Приборостроение».

Ключевые слова: изображение, алгоритмы обработки изображений, компьютерное зрение, OpenCV, Python, программа.

ISBN

Peter the Great St.Petersburg Polytechnic University

S.A.MOLODYAKOV

APPLICATION OF OPENCV FUNCTIONS IN COMPUTER VISION

(60 Python examples)

St.Petersburg
Polytechnic University Publishing House
2022

УДК 004.932

Reviewers:

S.M. Ustinov - Doctor of Technical Sciences, Professor of the Higher School of Software Engineering SPbPU

A.P. Lavrov - Doctor of Technical Sciences, Professor of the Higher School of Applied Physics and Space Technologies SPbPU

Application of OpenCV functions in computer vision (60 Python examples) / Molodyakov S.A. St.Petersburg Polytechnic University, 2022.- 296 p.

The monograph presents the issues of image processing using OpenCV functions. There are 60 examples of programs written in Python. The examples are divided into three levels of algorithms: simple low-level image processing, segmentation and high-level processing (recognition); and video processing. The presented examples are working programs written by students of the Higher School of Software Engineering of St. Petersburg Polytechnic University.

The monograph is intended for novice engineers and specialists in the field of design and application of video systems for image processing. Examples of programs allow you to simplify the study of computer vision algorithms and OpenCV library functions. The monograph can be useful to students who are trained in the areas of 09.03.01 "Computer Science and computer engineering", 09.03.04 "Software engineering", 12.03.01 "Instrument engineering".

Keywords: image, program, Python, OpenCV, image processing algorithms, computer vision.

ISBN

Оглавление

Введение. Особенности обработки изображений с применением функций OpenCV 9
Глава 1. Видеосистемы 12
1.1. Архитектуры видеосистем 12
1.2. Простая видеосистема 13
1.3. Видеосистема с распределенными камерами 14
1.4. Специализированная видеосистема 20
Глава 2. Простые низкоуровневые алгоритмы обработки изображений 23
2.1. Классификация алгоритмов 23
2.2. Простые операции над изображением 25
2.3. Пространственные методы 27
2.4. Среднегеометрический фильтр 30
2.5. Пространственный фильтр повышения резкости. Фильтр Лапласа 32
2.6. Частотные методы. Фильтр Баттерворта 34
2.7. Частотные методы. Фильтр Винера 38
2.8. Сглаживание и повышение резкости 43
2.9. Подмена пикселей 46
2.10. Изменение палитры цветов, псевдо раскраска 49
2.11. Определение HSV-кода пикселя в цветовом цилиндре 54
2.12. Определение преобладающих цветов на изображении с использованием кластеризации методом k-средних 57
2.13. Определение преобладающих цветов на изображении с использованием HSV палитры 59
2.14. Добавление водяного знака на видео 62
2.15. Алгоритм Retinex 64
2.16. Применение преобразования Хаара для подавления шумов на изображении 69
2.17. Пороговая обработка 73
2.18. Задача сопоставления изображений. Детекторы углов Харриса, LOG, DOG 77
2.19. Сравнение изображений и генерация картинки отличий 82
2.20. Фильтры в OpenCV 85
Глава 3. Сегментация изображений 90
3.1. Обзор методов сегментации изображений 90
3.2. Сегментация на основе цвета 92
3.3. Метод морфологического водораздела 97
3.4. Отделение объектов от фона. Алгоритмы Distance Transform и Watershed 104
3.5. Сегментация с использованием функции Canny 111
3.6. Алгоритмы сегментации, включая QuickShift 114
3.7. Алгоритм сегментации MeanShift 121
3.8. Сегментация с помощью нахождения краев и контуров 122
3.9. Суперпиксельная сегментация 126
3.10. Применение нейронной сети Mask RCNN для сегментирования 132
Глава 4. Алгоритмы высококоуровневой обработки изображений 140
4.1. Стабилизация видео 140
4.2. Стабилизация видео с помощью FFmpeg 142
4.3. Отслеживание объекта. Трекер 145
4.4. Отслеживание объекта, применение метода среднего сдвига 149
4.5. Определение жеста 152
4.6. Анализ частичного и полного оптического потока между кадрами изображения 156
4.7 Обобщенное преобразование Хафа и его применение для поиска объектов 159
4.8. Распознавание лиц с использованием каскадов Хаара 163
4.9. Определение улыбки 167
4.10. Распознавание и анимация глаз 169
4.11. Детектирование оставленных предметов с использованием imutils и dnn 175
4.12. Классификация изображений с использованием Inception V3 179
4.13. Пирамиды изображений 183
4.14. Поиск прямых и окружностей на изображении с помощью схемы голосования. Преобразование Хафа 186
4.15. Создание панорамных снимков 196
4.16. Скелетонизация 204

4.17. Детекторы областей IBR, MSER ... 206
4.18. Обратное преобразование перспективы ... 212
4.19 Проекции и преобразование Радона ... 215
4.20 Поиск совпадений между изображениями с помощью алгоритма полного перебора SIFT дескрипторов ... 217
Глава 5. Примеры использования алгоритмов обработки видео ... 221
5.1. Игра. Отслеживание объекта. Метод Гауссовой смеси ... 221
5.2. Рисование путем перемещения объекта перед видеокамерой ... 227
5.3. Сканер документов ... 233
5.4 Создание и чтение QR-кодов ... 239
5.5. Использование пирамиды изображений для уменьшения разрешения видео ... 244
5.6. Наложение моделей на изображение с веб-камеры с использованием технологий OpenCV и нейронных сетей (фреймворк dnn.py) ... 248
5.7. Детектирование медицинской маски на лице ... 251
5.8. Детектирование дорожных знаков на основе пороговой бинаризации ... 256
5.9. Определение дорожного знака ... 265
5.10. Распознавание автомобильных номеров ... 270
5.11. Замена фона в потоковом видео ... 273
5.12. Управление курсором мыши через жесты руки ... 276
5.13. Игра аэрохоккей ... 281
Заключение ... 287
Список литературы ... 288
Высшая школа программной инженерии (ВШПИ) СПбПУ ... 292

Студенты, которые принимали участие в написании содержания параграфов и разработке примеров программ

Параграф	Название	Автор
2.2.	Простые операции над изображением	Кемпи Елизавета
2.3.	Пространственные методы	Буровников Евгений
2.4.	Среднегеометрический фильтр	Воробьева Ольга
2.5.	Пространственный фильтр повышения резкости. Фильтр Лапласа	Афанасьев Егор
2.6.	Частотные методы. Фильтр Баттерворта	Коломыченко Артем
2.7.	Частотные методы. Фильтр Винера	Кичигин Юрий
2.8.	Сглаживание и повышение резкости	Мещеряков Григорий
2.11.	Определение HSV-кода пикселя в цветовом цилиндре	Подоба Александр
2.12.	Определение преобладающих цветов на изображении с использованием кластеризации методом k-средних	Черных Артем
2.13.	Определение преобладающих цветов на изображении с использованием HSV палитры	Сорин Николай
2.14.	Добавление водяного знака на видео	Письмерова Виктория
2.15.	Алгоритм Retinex	Ткаченко Александра
2.16.	Применение преобразования Хаара для подавления шумов на изображении	Назаренко Алёна, Щурихин Ярослав
2.17.	Пороговая обработка	Головин Михаил
2.18.	Задача сопоставления изображений. Детекторы углов Харриса, LOG, DOG.	Коновалов Александр
2.19.	Сравнение изображений и генерация картинки отличий	Ушаков Александр
2.20.	Фильтры в OpenCV	Шабинский Дмитрий
3.2.	Сегментация на основе цвета	Федорова София
3.4.	Отделение объектов от фона. Алгоритмы Distance Transform и Watershed	Хильченко Михаил
3.5.	Сегментация с использованием функции Canny	Бирюкова Марина
3.6.	Алгоритмы сегментации, включая QuickShift	Павлюченков Антон
3.7.	Алгоритм сегментации MeanShift	Симонова Евгения
3.8.	Сегментация с помощью нахождения краев и контуров	Казимиров Никита
3.9.	Суперпиксельная сегментация	Мейник Александр, Степанов Алексей, Филиппов Александр
3.10.	Применение нейронной сети Mask RCNN для сегментирования	Позднова Мария
4.2.	Стабилизация видео с помощью FFmpeg	Харитонов Лев
4.4.	Отслеживание объекта, применение метода среднего сдвига	Гусев Евгений
4.5.	Определение жеста	Громов Максим
4.6.	Анализ частичного и полного оптического потока между кадрами изображения	Гузов Артём
4.7.	Обобщенное преобразование Хафа и его	Касмынин Алексей

	применение для поиска объектов	
4.8.	Распознавание лиц	Дударов Александр
4.9.	Определение улыбки	Дмитриев Александр
4.10.	Распознавание и анимация глаз	Гречин Кирилл
4.11.	Детектирование оставленных предметов	Фомина Полина
4.12.	Классификация изображений. Inception V3	Кулачкова Анастасия
4.13.	Пирамиды изображений	Наумчик Анастасия
4.14.	Поиск прямых и окружностей на изображении с помощью схемы голосования. Преобразование Хафа	Ловкачева Юлия
4.15.	Создание панорамных снимков	Колесов Сергей
4.16.	Скелетонизация	Шутова Мария
4.17.	Детекторы областей (IBR, MSER)	Почетный Василий
4.18.	Обратное преобразование перспективы	Дзуцева Сабина
4.19.	Проекции и преобразование Радона	Дергунов Никита
4.20.	Поиск совпадений между изображениями с помощью алгоритма полного перебора SIFT дескрипторов	Весельский Сергей
5.1.	Игра. Отслеживание объекта. Метод Гауссовой смеси	Мухин Федор
5.2.	Рисование в OpenCV	Мэн Цзянин
5.3.	Сканер документов	Ершов Вадим
5.4.	Создание и чтение QR-кодов	Матус Арсений
5.5.	Использование пирамиды изображений для уменьшения разрешения видео	Драцкая Мария
5.6.	Наложение моделей на real-time изображение с веб-камеры с использованием технологий OpenCV и нейронных сетей (фреймворк dnn.py)	Степаненко Анастасия
5.7.	Детектирование медицинской маски на лице	Зайцев Егор
5.8.	Детектирование дорожных знаков на основе пороговой бинаризации	Андрианов Артемий
5.9.	Определение дорожного знака	Моданов Даниил
5.10.	Распознавание автомобильных номеров	Соколов Никита
5.11.	Замена фона в потоковом видео	Мишин Александр
5.12.	Управление курсором мыши через жесты руки	Столяров Алексей
5.13.	Игра аэрохоккей	Тарасов Никита

Введение.

Особенности обработки изображений с применением функций OpenCV

В настоящее время интенсивно развиваются все направления применения компьютерного зрения, создаются все более производительные и функционально насыщенные видеосистемы. Такое развитие базируется на создании все более высокопроизводительных узлов вычислительной техники, а также современных элементов фотоники.

Следует выделить следующие основные элементы видеосистем и связанной с ними вычислительной техники: КМОП- и ПЗС-фотоприемники, видеокамеры, видеопроцессоры на основе систем на кристалле или сигнальных процессоров, видеосистемы на кристалле, высокопроизводительные компьютеры и облачные системы. Эти элементы позволяют построить современную систему цифровой обработки изображений или компьютерного зрения. Но система компьютерного зрения является элементом видеосистемы, для работы которой в зависимости от решаемой задачи необходимы и другие узлы фотоники. К таким узлам относятся: лазеры, матрицы светодиодов, модуляторы света, фильтры, объективы, оптические волокна, волоконнооптические шайбы, высокопроизводительные цифровые интерфейсы и др.

На базе имеющихся новых аппаратных возможностей разрабатываются и применяются и новые алгоритмы анализа и обработки оптических сигналов и изображений. Эти алгоритмы решают следующие основные задачи: распознавание, идентификация, обнаружение,

восстановление 3D формы и др. [1-3]. Перечисленные алгоритмы являются высокоуровневыми алгоритмами. Они связаны с задачами искусственного интеллекта, соединением изображений конкретных объектов с картиной мира. Видеосистема же представляет данные для работы высокоуровневых алгоритмов. Предварительная обработка этих данных решается за счет применения алгоритмов низкоуровневой обработки видеоизображений. Эти алгоритмы являются универсальными для большинства видеосистем. Они обеспечивают в первую очередь улучшение качества видеокадров, управление цветовой палитрой, подчеркивание деталей, стабилизацию видео, синхронзацию с входным изображением, разрежение и др.

Разделение алгоритмов на уровни или модульную парадигму обработки изображений предложил Дэвид Кортни Марр [4]. Он определил три уровня: кроме низкоуровневых и высокоуровневых алгоритмов, он выделил средний уровень, связанный с задачами сегментирования или первоначальной классификации данных.

Считается, что низкоуровневые алгоритмы достаточно изучены. При этом имеется в виду алгоритмы, применяемые после фиксации потока кадров в цифровой памяти процессора. Однако имеется ряд вопросов, который требует рассмотрения. Во-первых, недостаточно исследованы и описаны низкоуровневые алгоритмы, реализуемые в фотоприемниках, то есть осуществляющие связь аналогового и цифрового пространств представления изображений. Во-вторых, среднеуровневые алгоритмы начинают объединяться с низкоуровневыми и для их реализации применяются общие устройства. В-третьих, современные вычислительные средства требуют разработки новых и модификации старых алгоритмов,

которые бы обеспечили параллельную и распределенную обработку данных.

В данной монографии рассматриваются перечисленные вопросы. Она является дальнейшим развитием вопросов, рассмотренных в монографиях «Проектирование специализированных цифровых видеокамер», «Низкоуровневая обработка изображений в компьютерном зрении». [5, 6].

В монографии основное внимание уделено представлению примеров программ, в которых используются функции библиотеки OpenCV (Open Source Computer Vision Library) [7, 8]. Программы позволяют упростить изучение алгоритмов компьютерного зрения и функций библиотеки OpenCV, их можно использовать при экспериментировании над изображениями. Интерес к функциям OpenCV объясняется и тем, что они могут быть реализованы не только в компьютерах, но и в системах программируемой логики. В библиотеке OpenCV реализовано более 500 алгоритмов компьютерного зрения. В монографии представлена реализация ряда алгоритмов, которые распределены на три уровня модульной парадигмы Дэвид Кортни Марр. Кроме того, отдельная глава выделена под примеры программ, которые работают с видео.

Глава 1. Видеосистемы

1.1. Архитектуры видеосистем

Можно перечислить множество областей, для которых требуется разрабатывать современные видеосистемы или системы потоковой обработки сигналов и изображений [9-19]. Данные в таких системах обрабатываются, и, может быть, представляются в темпе поступления сигналов. С точки зрения назначения видеосистем можно выделить два класса: системы обработки изображений, например, в роботах; и оптоэлектронные системы обработки сигналов, например, в радиоастрономии, локации, связи. Эти классы систем похожи наличием потоков кадров с фотоприемника, но отличаются элементами и применяемыми алгоритмами обработки. Видеосистемы обработки изображений в большинстве случаев представляют собой простые универсальные системы, которые включают объектив, фотоприемник и видеопроцессор. При рассмотрении алгоритмов обработки изображений с применение функций OpenCV рассматриваются именно такие видеосистемы. В случае оптоэлектронных систем обработки сигналов и изображений видеосистема является элементом более сложной оптоэлектронной системы, и алгоритмы работы видеосистемы определяются решаемой в оптоэлектронной системе задачей. Так в оптоэлектронной системе, созданной для обработки и регистрации сигналов радиоизлучения пульсаров, реализуются переключаемые во времени режимы временной задержки и накопления (ВЗН) и кадрового накопления, а также некоторые режимы попиксельной обработки [14].

С точки зрения построения видеосистем обработки изображений можно выделить следующие схемы: простая видеосистема (с одной

видеокамерой), видеосистема с распределенными видеокамерами и специализированная видеосистема (со специализированной видеокамерой).

1.2. Простая видеосистема

Обобщенная структура простой видеосистемы потоковой обработки сигналов и изображений приведена на рис. 1.1. Она включает оптический узел, фотоприемник (ФП), видеопроцессор и компьютер [5, 20, 21]. В качестве оптического узла в зависимости от системы могут быть элементы проецирования изображения или оптический вычислитель (оптический процессор). Если в первом случае обрабатывается изображение, например для задачи распознавания, то во втором случае обрабатывается входной сигнал $U_{\text{ВХ}}$, который в процессе вычислений преобразуется в оптическую форму и также проецируется на ФП. Могут использоваться фотоприемники разных типов и различного разрешения. Особый интерес вызывают фотоприемники, которые позволяют обрабатывать сигналы на самом кристалле фотоприемника. Такая буферная обработка обеспечивает существенное сжатие информации до ввода ее в цифровые вычислители. Буферную обработку можно проводить как на приборах с зарядовой связью (ПЗС-фотоприемники), так и на фотоприемниках на комплементарных структурах метал-оксид-полупроводник (КМОП-фотоприемники) [20, 22, 23]. Видеопроцессоры могут быть построены на программируемой логике, цифровых сигнальных процессорах (ЦСП) и системах на кристалле [5]. Заканчивается обработка в компьютере. Функции OpenCV могут использоваться как в видеопроцессоре, так и в компьютере.

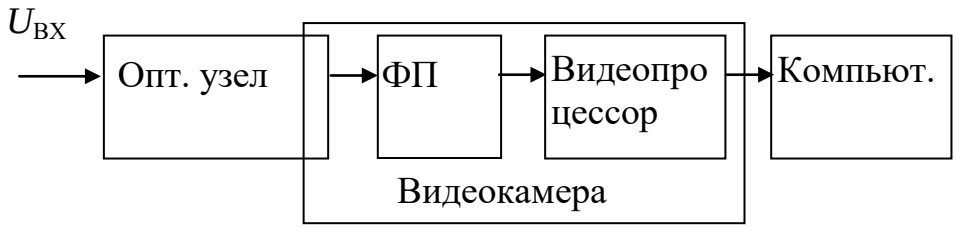


Рис. 1.1. Обобщенная схема видеосистемы.

На рисунке выделена видеокамера, которая может включать фотоприемник и видеопроцессор. Известны видеосистемы на кристалле, которые представляют собой микропроцессорное исполнение видеокамеры. Видеокамера является законченным элементом видеосистемы, поэтому часто она дополнительно включает входной объектив и интерфейс связи с компьютером.

Особенность видеосистем заключается в том, что в ряде применений не хватает производительности компьютера и других цифровых устройств. Приходится использовать различные алгоритмы распределенной и параллельной обработки. Причем вид алгоритма во многом определяется теми возможностями, которые предоставляет ФП для предварительного сжатия (буферной обработки) данных. Поэтому важно знать и уметь использовать возможности ФП при проектировании систем. Наличие разнородных элементов в видеосистемах и распределенной по узлам обработки вызывает необходимость совместного или сопряженного проектирования оптического узла, ФП, цифрового узла и программного обеспечения (ПО) [22, 24].

1.3. Видеосистема с распределенными камерами

Видеосистема с распределенными камерами представляют собой систему, включающую несколько камер, связанных решением общей задачи. Различают две схемы соединения камер: централизованная (радиальная) и сетевая (рис. 1.2, 1.3). В первом случае камеры

подключены к единому устройству, например, компьютеру или серверу, а во втором случае связаны между собой, например, через сеть. Наиболее интересна сетевая архитектура видеосистемы. Она требует использования интеллектуальных обычно IP-камер, которые самостоятельно могут решать различные задачи, а также обмениваться между собой информацией. Это новый подход построения распределенных видеосистем, связанный с использованием высоких вычислительных и коммуникационных мощностей камер [25, 26].

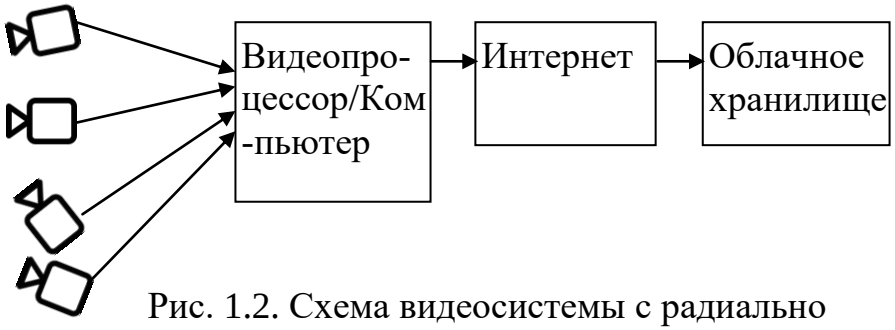


Рис. 1.2. Схема видеосистемы с радиально распределенными камерами.

Сеть, объединяющую множество камер, можно рассматривать как Smart-видеосистему. Она может динамически перенастраивать свою конфигурацию для выполнения различных задач, менять режимы выделения ограниченных ресурсов, обмениваться данными и координировать функционирование разнотипных устройств.

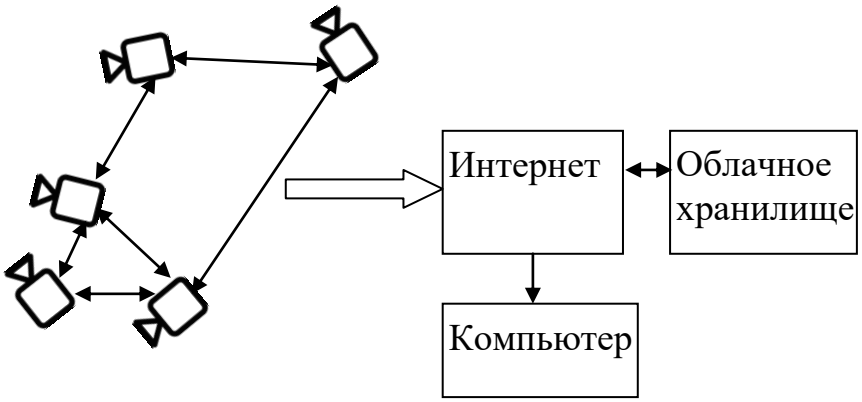


Рис. 1.3. Схема видеосистемы с сетевым соединением камер.

Сетевая схема позволяет использовать новую стратегию распределения интеллектуальных функций. Она позволяет преодолеть ограничения, которые имеют централизованные архитектуры. Основная обработка видео осуществляется самими сетевыми камерами. Такую обработку видео называют обработкой "на переднем крае". Сетевая архитектура требует наименьшей полосы пропускания, поскольку камеры сами производят анализ и посылают только ту информацию и видео, которые требуются. Это помогает значительно снизить затраты и сложность системы по сравнению с централизованной архитектурой и полностью устранить ее недостатки.

Например, если камера имеет детектор движения, то осуществляется пересылка и сохранение для последующего анализа только те кадры, в которых имеется движение. Для функций специального логического анализа, где требуется только передача данных, а не видео, таких как подсчет посетителей или автоматическое распознавание номерных знаков, выполнение этих функций самой камерой приводит к значительному снижению нагрузки на сеть, поскольку камера может вычленять и пересылать только требующиеся данные.

Благодаря интеграции устройств "на переднем крае" с системой управления видео и равномерному распределению нагрузки между различными участками сети, интеллектуальные решения для видео позволяют практически неограниченное расширение системы, увеличивают ее гибкость и экономичность по сравнению с централизованными решениями. Снижаются требования к производительности серверов. Серверы, которые обычно производят полную обработку лишь нескольких потоков, могут обрабатывать более

сотни потоков, если часть процесса обработки уже выполнена самой камерой.

В интеллектуальных сетях камер следует выделить два этапа работы: самонастройка, реконфигурируемость. Самонастройка происходит при начальной стадии работы камер. Реконфигурируемость связана с перестройкой сети в процессе ее работы.

Можно выделить пять основных функций процедуры самонастройки для систем видеонаблюдений [26]:

1. Настройка каждой из камер с учетом их физического положения (калибровки), ресурсов энергопотребления (батареи), чувствительности (типа датчика и поля зрения), вычислительных ресурсов (процессора и памяти) и телекоммуникационного протокола.

2. Настройка сети взаимодействия камер с точки зрения обмена информацией между ними и совместного использования зон обзора. Описываются соседи для каждой из камер и каналы взаимодействия. Возможности обмена информацией могут меняться, поскольку характеристики камер, их число и пропускная способность каналов связи с течением времени также меняются. Любой протокол, с помощью которого осуществляется управление связью и обработкой данных, также должен учитывать различные аспекты. Сюда относятся политики оптимального распределения имеющейся пропускной способности, устойчивость к ошибкам передачи, маршрутизация информации об эффективности энергопотребления и поддержка связи между разнотипными устройствами.

3. Настройка расположения камер в среде наблюдения. Описываются наблюдательные позиции, характеризующие зоны обзора камер, а также

любые статические и движущиеся препятствия, которые могут блокировать поле зрения.

4. Настройка механизма принятия решений, связанного с конкретными задачами. Эти решения обеспечивают выполнение заданий — в частности, распознавание появления и определение положения движущихся целей. Операционные решения помогают организовать обмен информацией между камерами и их координацию, в частности планирование процедур анализа видео. Элемент автоматической настройки конфигурации задачи включает ресурсные ограничения (стоимость энергии и вычислений), а также обеспечивает сбалансированность распределения нагрузки между камерами. Сбалансированность, в свою очередь, требует декомпозиции задач в зависимости от их сложности и потребляемых ресурсов. Распознавание поведения, например, может разбиваться на определение объекта, его функций и пространственно-временных характеристик. После этого каждая из выделенных подзадач поручается какой-либо наиболее подходящей камере. Число задач, например, может превышать количество камер, а при распределении последних приходится решать еще и задачи, связанные с адаптацией и сменой направленности множества зон обзора.

5. Настройка производительности камер. Показатель производительности позволяет оценить успех стратегии реконфигурации. В качестве критерия оптимального состояния сети могут выступать и другие показатели: частота захваченных и обрабатываемых кадров, разрешение изображения, степень сжатия.

Сети интеллектуальных камер должны быть готовы противостоять ошибкам, ограничениям пропускной способности и меняющимся временным параметрам. При перемещении камер, их отключении в целях

экономии электроэнергии, подключении к сети и выходе из нее в сетевой структуре происходят определенные изменения. Возможности реконфигурации должны предусматривать адаптацию к возникновению обстоятельств, которые не всегда известны заранее. Например, от сети интеллектуальных камер может потребоваться обновление конфигурации при одновременной съемке горящего здания и отслеживании действий пожарных и спасающихся жильцов.

Реконфигурируемость сети состоит в непрерывной (регулярной) оценке топологии, самокалибровке, распределении ресурсов и задач. Эффективность реконфигурации зависит от способности сети к самостоятельной калибровке и анализа топологии при ее определении и обновлении. Например, изменение зоны обзора камеры требует применения технологий активного видеонаблюдения, позволяющих управлять перемещением камер и контролировать их размещение. При перемещении любой из камер топология сети меняется, что может потребовать перемещения и ее соседей.

Сеть камер может включать разнородные камеры, связанные различными интерфейсами. Например, на рис.1.4 представлена сеть IP-камер, связанных через Ethernet и радиоканал с сервером. Камеры соединены с портами 100 Мбит/с коммутатора №2. Если максимальный поток с одной камеры составляет 8 Мбит/с, то пиковый поток семи камер составляет 56 Мбит/с. Цифровой поток с этой скоростью должен непрерывно поступать на сетевой видеосервер. Реальная скорость передачи в канале 100 Мбит/с редко превышает 50 – 60 Мбит/с. Поэтому используется второй быстродействующий коммутатор. Этот коммутатор оснащен портами 1 Гбит/с, к которым подключен сетевой видеосервер, компьютер оператора и роутер.

Рис.1.4. Сеть IP-камер.

1.4. Специализированная видеосистема

Специализированная видеосистема предназначена для решения определенных задач, для которых не справляется традиционная видеосистема. Обычно специализацию задает наличие видео процессора или нестандартных фотоприемников или фотоприемников, работающих нестандартно. Особое функционирование может быть реализовано как на аппаратном, так и на программном уровне встроенного ПО.

Примером специализированной видеосистемы является система для оптической астрономии, в которой используется не один фотоприемник, а целое поле фотоприемников, специально совмещенных между собой [27].

Примером видеосистемы, в которой используется специальный режим - режим временной задержки и накопления (ВЗН), является видеосистема для решения задач наблюдения сигналов в радиоастрономии [28-30]. Другими областями, в которых используются специализированные видеосистемы, являются: системы безопасности, авиационные и космические приложения, распознавание текста и штриховых кодов, биометрия и др.

Автор участвовал в ряде разработок систем со следующими основными направлениями использования: радиоастрономия, связь, радиомониторинг воздушного пространства, обработка сигналов радиолокаторов с синтезируемой апертурой антенны (РСА) [28-33].

Приведем пример специализированной видеосистемы, которая позволяет обрабатывать и регистрировать сигналы радиоизлучения пульсаров. Такая система работала на радиотелескопе РТ-64 в Калязине.

Эквивалентная схема приемного комплекса радиотелескопа с видеосистемой представлена на рис. 1.5. На ней показаны следующие основные элементы: СВЧ-приемник, акустооптический (АО) процессор, ВЗН-ФПЗС (режим временной задержки и накопления - ВЗН), АЦП с усилителем и цифровой сигнальный процессор (ЦСП). В системе могут использоваться 1D или 2D АО-процессоры, выполняющие спектральное или спектрально-корреляционное преобразование сигнала. В первом случае применяется линейный, а во втором – матричный ВЗН-ФПЗС. Оптический сигнал, соответствующий спектру входного радиоизлучения пульсара, с выхода АО-процессора перемещается по полю фотоприемника с постоянной скоростью V_S, связанной с типом (мерой дисперсии) наблюдаемого пульсара. Период сигнала определяется периодом принимаемого радиоизлучения. Скорость перемещения оптического сигнала в рассматриваемой системе составляет $10^5 - 10^6$ элемента в секунду (через 1-16 мкс сигнал переходит на следующий элемент фотоприемника) [14].

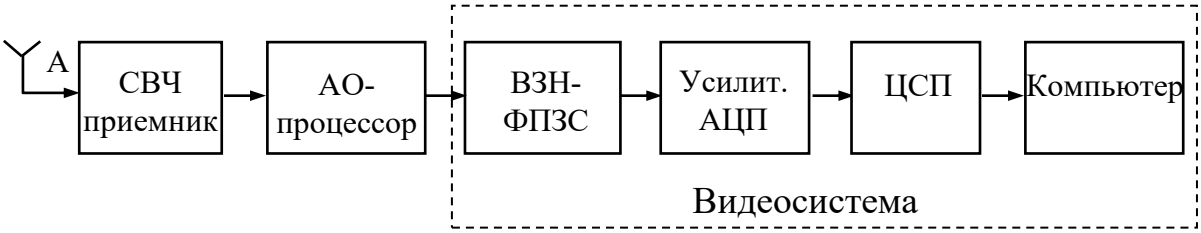


Рис. 1.5. Эквивалентная схема приемного комплекса радиотелескопа с видеосистемой

При использовании 2D АО-процессора оптический сигнал представляет собой динамическую интерферограмму [22]. Для обработки и накопления перемещающихся сигналов не удается использовать кадровый режим накопления, так как частота вывода кадров в этом случае будет превосходить частоту 1 ГГц, которую не поддерживают ПЗС-фотоприемники. В результате мы используем ВЗН-режим, в котором зарядовые пакеты под действием управляющих сигналов перемещаются электронным способом вдоль апертуры ФПЗС от одного края к другому подобно непрерывной цепочке (конвейеру).

Время накопления элементарного кадра или минимальный период дискретизации оптического сигнала определяется временем вывода одной строки T_L. Полное время интегрирования оптического сигнала в матрице равно $T_{INT}=M\cdot T_L$. В результате накапливаемый сигнал Q_Σ в M раз больше, чем сигнал при одном элементарном кадре накопления ФПЗС. Для уменьшения времени T_L используют несколько узлов вывода зарядовых пакетов, так в ВЗН-ФПЗС типа 21241 их 16.

Реализовать ВЗН-режим можно на линейных и матричных ФПЗС как специальной ВЗН-структуры, так и традиционных кадровых ФПЗС при специальном управлении [22].

Глава 2. Простые низкоуровневые алгоритмы обработки изображений

2.1. Классификация алгоритмов

Известны несколько принципов классификации алгоритмов обработки изображений. Основной принцип связан с модульной парадигмой обработки изображений Дэвида Кортни Марра. В этом случае выделяют три уровня алгоритмов: попиксельной обработки, сегментирования и распознавания. Причем под попиксельной обработкой понимают все алгоритмы, связанные с улучшением кадров изображений. Алгоритмы попиксельной обработки можно определить в качестве низкоуровневых алгоритмов, так как они реализуются на низком уровне программирования в фотоприемниках (возможно в программируемой логике) или в видеопроцессорах (возможно ассемблер или другой низкоуровневый язык). При реализации таких алгоритмов требуется минимальное количество действий или команд, так как они работают в темпе поступления кадров изображений в видеосистемах.

С точки зрения реализации низкоуровневых алгоритмов можно предложить классификацию, представленную в табл. 2.1. Они разделены на стадии применения и могут быть реализованы как в аналоговом, так и в цифровом виде в цифровых процессорах и в программируемой логике. Высокоуровневые алгоритмы сегментирования и распознавания можно проводить на стадии покадровой обработки. В таблице представлены алгоритмы обработки изображений. Алгоритмы низкоуровневой обработки для оптоэлектронных систем представлены в работах [31-33].

Табл. 2.1. Классификация низкоуровневых алгоритмов обработки изображений в видеосистемах.

Стадия применения	Название алгоритмов, используемых в цифровых видеосистемах
Управление фотоприемником	Автоматическая экспозиция (Auto exposure AE). Автобаланс белого (Auto white balance AWB). Детектирование плавания сигнала (Flicker avoid).
Попиксельная обработка	Компенсация черного (Auto black reference ABR). Программирование аналогового усиления. Программирование времени экспозиции. Выбор разрядности АЦП. Контроль насыщения. Преобразование Bayer RGB.
Построковая обработка	Интерполяция цвета. Увеличение/уменьшение. Коррекция дефектов. Коррекция цвета. Гамма коррекция. Компенсация искажений в линзе. Объединение отсчетов(Binning). Подавление шумов.
Покадровая обработка	Контроль резкости (Sharpness). Подчеркивание границ. Изменение формата, сжатие.
Управление режимом	Регистрация одного кадра. Потоковое видео. Переключение памяти данных (flash)

Низкоуровневые алгоритмы разделены на пять стадий. Попиксельная обработка реализуется после получения значений одного или нескольких соседних пикселей. Построковая обработка реализуется после получения значений всей строки пикселей. Например, так происходит в КМОП-фотоприемниках. Покадровая обработка реализуется после получения значений всех пикселей кадра. Так происходит в видеосистемах, в которых используется видеокамера, и кадры в компьютер следуют один за другим. Выделены стадии управления фотоприемником и управления режимом работы видеосистемы.

Другая классификация алгоритмов предложена Ричардом Шелиски [1]. Он выделил простые алгоритмы (линейная фильтрация, пирамиды, геометрические преобразования …), более сложные (сопоставление, обнаружение функций, сшивание изображений …), сегментацию и

обнаружение/ распознавание. Отдельно выделяются алгоритмы работы с видео. Для большинства рассмотренных алгоритмов имеются поддерживающие или реализующие их функции из библиотеки OpenCV. Поэтому на основе классификации Шелиски проведено разделение примеров реализации ряда алгоритмов на главы.

В качестве основного языка программирования выбран язык Python [34, 35]. Он в настоящее время является основным языком для изучения функций множества библиотек программ, как для компьютерного зрения, так и для задач с применением методов машинного обучения [36, 37].

2.2. Простые операции над изображением

Приведем примеры использования функций OpenCV для выполнения простых операций над изображением. Такими операциями являются: обрезка изображения, изменение размера изображения, поворот изображения, добавление линий на изображение. добавление текста на изображение.

Используемые функции

cv2.warpAffine	Вращение
cv2.resize	Изменение размера
cv2.line	Добавление линии
cv2.putText	Добавление текста
cv2.waitKey	Ожидание нажатия клавиши
cv2.imread	Считывание исходного изображения
cv2.imShow	Показ изображения

Листинг программы

```
import cv2

picture = cv2.imread("summer1.jpg")
cv2.imshow("Начальное изображение", picture )
cv2.waitKey(-1); cv2.destroyAllWindows()

def vP(picture, x):
    cv2.namedWindow(x, cv2.WINDOW_NORMAL); cv2.imshow(x, picture)
    cv2.waitKey(-1); cv2.destroyAllWindows()
vP(picture [15:480, 480:1900], "Обрезание изображения")

pict = cv2.imread("summer1.jpg")
```

```
d = (int(pict.shape[1] * 40 / 100), int(pict.shape[0] * 40 / 100))
vP(cv2.resize(pict, d, interpolation = cv2.INTER_AREA), "Изменение размера на 40
%")

(h, w, d) = pict.shape; c= (w // 2, h // 2)
vP(cv2.warpAffine(pict, cv2.getRotationMatrix2D(c, 45, 1), (w, h)), "Поворот на 45
градусов")

o = pict.copy()
cv2.line(o, (810, 810), (110, 810), (20, 15, 255), 8)
vP(o, "Линия")

o = pict.copy()
cv2.putText(o, "YES!!!", (400, 400),cv2.FONT_HERSHEY_SIMPLEX, 9, (20, 205, 215),
25)
vP(o, "Текстом")
```

Результат работы программы

Начальное изображение

Изменение размера

Обрезка изображения

Поворот на 45 градусов

Добавление текста на изображение

Добавление линий на изображение

2.3. Пространственные методы

Рассмотрим пространственные фильтры.

MedianBlur. В медианном фильтре мы выбираем скользящее окно, которое будет перемещаться по всем пикселям изображения. Здесь мы берем все значения пикселов, которые попадают в фильтр некоторой размерности, и принимаем медиану этих значений. Результат работы будет присвоен центральному пикселю.

Например, фильтр размерности 3 на 3 будет со следующими значениями после размещении его на изображении. Чтобы рассчитать медиану, которая считается средним значением отсортированной последовательности чисел, мы отсортируем все элементы матрицы и найдем среднее.

43	65	29
17	57	98
63	84	59

Сортировка даст следующие значения:

```
17 29 43 57 59 63 65 84 98
```

Чтобы найти среднее, посчитаем количество всех чисел, добавим к этому значению 1 и разделим на 2. Если мы применим данный алгоритм к нашему случаю, то результатом будет число 59. Это и будет новым значением пиксела.

Filter2D and mean filter. Так называемый средний фильтр заменяет текущее значение пиксела на среднее значение окрестности. Таким образом, при средней фильтрации каждый пиксель изображения будет заменен средним значением его соседей и самого пиксела. Это приводит к удалению ряби, сглаживанию шума, цветовому сглаживанию всего

изображения. Рассмотрим пример, чтобы точно разобраться в работе данного алгоритма:

Пусть у нас имеется набор пикселов со следующей матрицей, и мы будем применять к нему фильтр средних значений:

7	9	23
76	91	7
64	90	32

Теперь новым значением для центрального пиксела будет 44, вместо 91.

Erode. При эрозии ядро также играет основную роль в свёртывании изображения. Если мы рассмотрим все пикселы в двоичном представлении, то каждый рассматриваемый пиксел будет равен 1, если в ядре будут только 1. Иначе этот пиксел будет 0. В зависимости от размера ядра, все пикселы у границы отбрасываются, поэтому толщина и размер объекта переднего плана может изменяться (обычно уменьшается).

Эрозия очень полезна для удаления мелкой белой ряби, мелких связанных объектов и т.д.

Список используемых функций

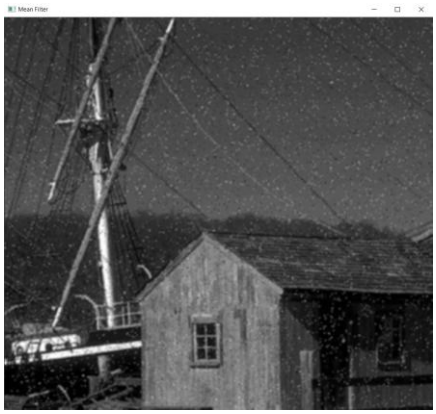
Название	Краткое описание
cv2.imshow()	Метод по отображению изображения в отдельном окне. Размер окна автоматически подстраивается под размер изображения
cv2.imwrite()	Метод для сохранения изображения в соответствии с указанным форматом в текущем рабочем каталоге.
cv2.imread()	Метод для чтения изображения. Возвращает 2D или 3D матрицу (в зависимости от количества цветовых каналов).
cv2.waitKey()	Метод ожидания нажатия клавиши. Возвращает код нажатой клавиши
cv2.destroyAllWindows()	Уничтожает все открытые окна
cv2.medianBlur()	Размывает изображение с помощью медианного фильтра.
cv2.filter2D()	Позволяет изменить изображение в соответствии с указанным ядром
cv2.erode()	Функция размывает исходный образ с помощью указанного элемента структурирования, который определяет форму пиксельной окрестности

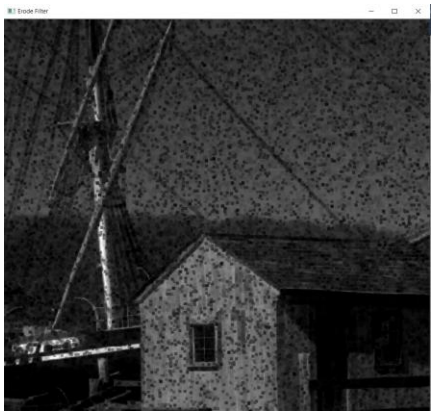
Листинг программы

```
import cv2
import numpy as np


def improve_image_with_median_filter(image_for_median):
    processed_image = cv2.medianBlur(image_for_median, 7)
    cv2.imshow('Median Filter', processed_image)
    cv2.imwrite('median_processed_image.png', processed_image)

def improve_image_with_mean_filter(image_for_mean):
    kernel = np.ones((5, 5), np.float32) / 25
    processed_image = cv2.filter2D(image_for_mean, -1, kernel)
    cv2.imshow('Mean Filter', processed_image)
    cv2.imwrite('mean_processed_image.png', processed_image)


def improve_image_with_erode(image_for_erode):
    kernel = np.ones((5, 5), np.uint8)
    processed_image = cv2.erode(image_for_erode, kernel, iterations=1)
    cv2.imshow('Erode Filter', processed_image)
    cv2.imwrite('erode_processed_image.png', processed_image)


first_image = cv2.imread('first.jpg')
second_image = cv2.imread('second.jpg')
third_image = cv2.imread('third.jpg')
improve_image_with_median_filter(second_image)
improve_image_with_mean_filter(second_image)
improve_image_with_erode(second_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Примеры применения программы

Исходное изображение





2.4. Среднегеометрический фильтр

Фильтр среднего геометрического — это фильтрация изображения, предназначенная для сглаживания и уменьшения шума. Он основан на математическом среднем геометрическом. Фильтр среднего геометрического наиболее широко используется для фильтрации гауссовского шума. В общем, он помогает сгладить изображение с меньшими потерями данных, чем фильтр среднего арифметического.

Изображение, восстановленное с использованием среднегеометрического фильтра G (x, y), задается выражением:

$$G(x,y) = \left[\prod_{i,j \in S} S(i,j)\right]^{1/m*n}$$

Формула 1

Где S (x, y) - исходное изображение, а m и n - размерность маски фильтра.

Каждый пиксель выходного изображения в точке (x, y) задается произведением пикселей в маске среднего геометрического, возведенным в степень 1 / m*n.

Например, при использовании размера маски 3 на 3 пиксель (x, y) в выходном изображении будет произведением S(x, y) и всех 8 окружающих его пикселей в степени 1/9.

Используя следующее исходное изображение с пикселем (x, y) в центре:

Дает результат: (5*16*22*6*3*18*12*3*15) ^(1/9) = 8.77.

Программа преобразует цветное изображение с помощью алгоритма среднегеометрической фильтрации. Изображение разделяется по цветам RGB, и с каждым получившимся одноканальным массивом осуществляются преобразования по Формуле 1. Получившиеся преобразованные массивы объединяются и выводятся.

Таблица использованных функций

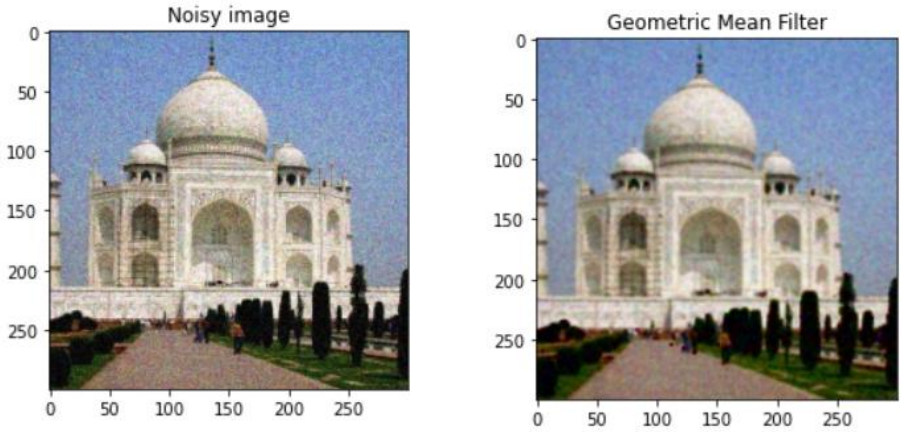
cv2.imread	Функция imread загружает изображение из указанного файла и возвращает его. Если изображение не может быть прочитано (из-за отсутствия файла, неправильных разрешений, неподдерживаемого или недопустимого формата), функция возвращает пустую матрицу
cv2.copyMakeBorder	Используется для создания границы вокруг изображения
cv2.split	Разделяет многоканальный массив на несколько одноканальных
cv2.merge	Объединяет вектор одноканальных массивов в один многоканальный массив
cv2.cvtColor	Используется для преобразования изображения из одного цветового пространства в другое
np.prod	Возвращает произведение элементов массива по заданной оси
np.zeros	Возвращает новый массив заданной формы и типа, заполненный нулями.
plt.imshow	Отображает данные в виде изображения
plt.title	Задает название для изображения

Листинг программы

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

def GeometricMeanAlogrithm(image):
    new_image = np.zeros(image.shape)
    image = cv2.copyMakeBorder(image, 1, 1, 1, 1, cv2.BORDER_DEFAULT)
```

```
    for i in range(1, image.shape[0] - 1):
        for j in range(1, image.shape[1] - 1):
            roi = image[i - 1:i + 2, j - 1:j + 2]
            roi = roi.astype(np.float64)
            p = np.prod(roi) #product of array elements over a given axis.
            new_image[i - 1, j - 1] = p ** (1 / (roi.shape[0] * roi.shape[1]))
    return new_image.astype(np.uint8)

def rgbGeometricMean(image):
    r,g,b = cv2.split(image)
    r = GeometricMeanAlogrithm(r)
    g = GeometricMeanAlogrithm(g)
    b = GeometricMeanAlogrithm(b)
    return cv2.merge([r,g,b])

img = cv2.imread('img.jpg')
image = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
plt.imshow(image)
plt.title("Noisy image")
plt.show()

newImg = rgbGeometricMean(image)
plt.imshow(newImg)
plt.title("Geometric Mean Filter")
plt.show()
```

Пример работы программы



2.5. Пространственный фильтр повышения резкости. Фильтр Лапласа

Оператор Лапласа является производным оператором, который используется для поиска ребер в изображении. Это производная маска второго порядка. В этой маске у нас есть еще две классификации: одна — положительный оператор Лапласа, а другая — отрицательный оператор Лапласа.

Поскольку оператор Лапласа является производной второго порядка, он подчеркивает прерывистую полутоновую часть пикселя изображения, не подчеркивая при этом непрерывное значение серой шкалы. Это создаст черный фон с четкой серой границей, но недостаточно функций.

После записи оператора Лапласа в качестве маски фильтра его работа аналогична другим операциям пространственной фильтрации. Перемещайте маску фильтра построчно на исходном изображении, а затем значение в маске умножается на пиксели, которые совпадают друг с другом, а затем суммируется и присваивается пикселям, которые совпадают с центром маски. Последние строки и столбцы изображения нельзя использовать для вышеуказанных операций. Пикселю присваивается значение ноль, и получается результат лапласовской операции.

0	1	0
1	-4	1
0	1	0

Форма маски положительного оператора Лапласа

0	-1	0
-1	4	-1
0	-1	0

Форма маски отрицательного оператора Лапласа

Использованные функции

Название	Краткое описание
cv2.imshow()	Метод по отображению изображения в отдельном окне. Размер окна автоматически подстраивается под размер изображения
cv2.nameWindow()	Метод дает название отображаемому окну
cv2.imread()	Метод для чтения изображения. Возвращает 2D или 3D матрицу (в зависимости от количества цветовых каналов).
cv2.waitKey()	Метод ожидания нажатия клавиши. Возвращает код нажатой клавиши
cv2.destroyAllWindows()	Уничтожает все открытые окна
cv2.Laplacian()	Использование пространственного фильтра с повышением резкости
cv2.GaussianBlur()	Использует фильтр Гаусса, который представляет собой фильтр нижних частот, который удаляет высокочастотные

	составляющие
cv2.cvtColor()	Конвертирует цвета изображения
cv2.convertScaleAbs()	Масштабирует, вычисляет абсолютные значения и преобразует результат в 8-битный формат

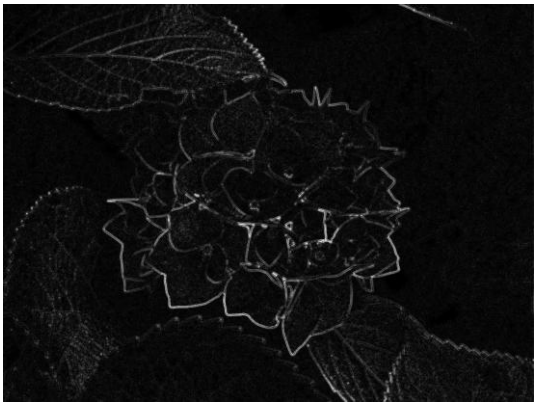
Листинг программы

```
import cv2 as cv

ddepth = cv.CV_16S
kernel_size = 3
window_name = "Laplace"
window_name_original = "Original"
imageName = '012.jpg'
src = cv.imread(cv.samples.findFile(imageName), cv.IMREAD_COLOR)

cv.imshow(window_name_original, src)

src = cv.GaussianBlur(src, (3, 3), 0)
src_gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
cv.namedWindow(window_name, cv.WINDOW_AUTOSIZE)

dst = cv.Laplacian(src_gray, ddepth, ksize=kernel_size)
abs_dst = cv.convertScaleAbs(dst)

cv.imshow(window_name, abs_dst)

cv.waitKey(0)
cv.destroyAllWindows()
```

Результаты работы

Исходное изображение

Полученное изображение



2.6. Частотные методы. Фильтр Баттерворта

Преобразование изображений традиционно ведется в пространственной области, где маски фильтров и алгоритмы преобразования применяются непосредственно к матрице изображения.

Иными словами, осуществляется переход от непосредственно изображения к его спектру с помощью некоторого набора базисных функций. Самым распространенным способом является Фурье-преобразование.

Основными видами используемых фильтров являются:

- низкочастотный фильтр, ослабляющий высокие частоты, одновременно пропуская низкие
- высокочастотный фильтр, обладающий противоположными свойствами.

Низкие частоты Фурье-преобразования отвечают за возникновение превалирующих значений яркости на гладких участках изображения, в то время как высокие частоты отвечают преимущественно за контуры и шум. После применения низкочастотной фильтрации изображение по сравнению с исходным изображением содержит меньше резких деталей.

В случае же применения высокочастотной фильтрации на изображении уменьшаются изменения яркости в пределах больших гладких областей и выделяются переходные зоны быстрого изменения яркости, то есть контуры изображения. Как правило, такое изображение обладает большей резкостью по сравнению с исходным.

Модель фильтрации изображения в частотной области в общем виде представляет собой следующее равенство:

$$G(u,v) = H(u,v) \cdot F(u,v)$$

где:
$F(u,v)$ – Фурье-образ изображения, подлежащий фильтрации
$H(u,v)$ – передаточная функция фильтра
$G(u,v)$ – итоговая функция

Рассмотрим фильтры Баттерворта низких и высоких частот.

Передаточная функция низкочастотного фильтра Баттерворта порядка n на расстоянии D0 от начала координат задается формулой:

$$H(u,v) = \frac{1}{1 + (\sqrt{2} - 1) \cdot (D(u,v)/D_0)^{2n}}$$

К преимуществам низкочастотных фильтров Баттерворта относится намного меньшее проявление нежелательных эффектов размытия и появления ложных контуров по сравнению с идеальными низкочастотными фильтрами. С увеличением порядка низкочастотного фильтра Баттерворта возрастает проявление эффектов размытия. Принято считать, что низкочастотный фильтр Баттерворта второго порядка является оптимальным с точки зрения компромисса между эффективностью низкочастотной фильтрации и приемлемым уровнем проявления ложных контуров и общего размытия изображения.

Передаточная функция высокочастотного фильтра Баттерворта порядка n на расстоянии D0 от начала координат задается формулой:

$$H(u,v) = \frac{1}{1 + (\sqrt{2} - 1) \cdot (D_0 / D(u,v))^{2n}}$$

Фильтры Баттерворта высоких частот приводят к гораздо меньшим искажениям границ объектов, чем идеальные высокочастотные фильтры Баттерворта. С увеличением порядка высокочастотного фильтра Баттерворта искажения границ объектов заметно увеличиваются.

Используемые функции

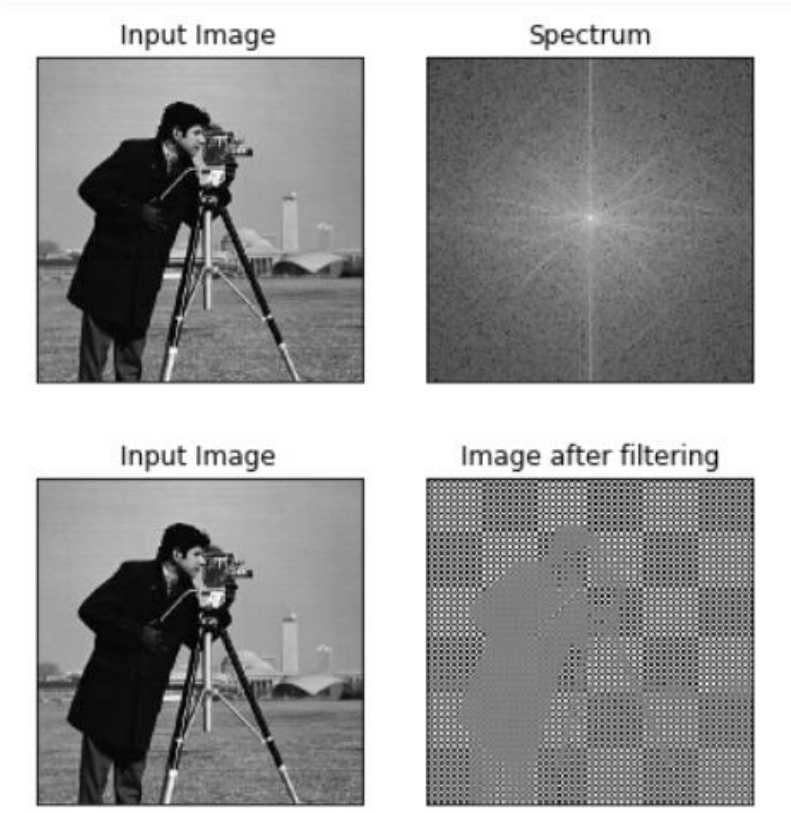
np.fft.fft2() – выполняет двумерное дискретное преобразование Фурье
np.fft.fftshift() – сдвигает компонент нулевой частоты в центр спектра
np.fft.ifft2() – выполняет обратное двумерное дискретное преобразование Фурье
np.fft.ifftshift() – выполняет действие, обратное *np.fft.fftshift()*

Листинг программы

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Функция фильтра Баттерворта
def butterworth_filter(size, cutoff, n):
    b = np.sqrt(2) - 1
    rows = size[0]
```

```
    columns = size[1]

    x =  np.linspace(-0.5, 0.5, rows)
    y =  np.linspace(-0.5, 0.5, columns)

    radius = np.sqrt((x**2)[np.newaxis] + (y**2)[:, np.newaxis])
    filt = 1 / (1.0 + b * (cutoff / radius)**(2*n))

    return filt

# Считываем изображение
img = cv2.imread('cameraman.jpg', 0)

# Выполняем дискретное преобразование Фурье
img_fft = np.fft.fft2(img)

# Получаем амплитудный спектр
img_shift = np.fft.fftshift(img_fft)
img_amp = 20*np.log(np.abs(img_shift))

# Выводим исходную картинку и амплитудный спектр
plt.subplot(121)
plt.imshow(img, cmap = 'gray')
plt.title('Input Image')
plt.xticks([])
plt.yticks([])

plt.subplot(122)
plt.imshow(img_amp, cmap = 'gray')
plt.title('Spectrum')
plt.xticks([])
plt.yticks([])
plt.show()

# Применяем фильтр
filt = butterworth_filter(img.shape[1::-1], 70, 2)
img_filt = img_fft * filt

# Конвертируем обратно в изображение
img_new = np.real(np.fft.ifft2(np.fft.ifftshift(img_filt)))

# Выводим исходную картинку и картинку с наложенным фильтром
plt.subplot(121)
plt.imshow(img, cmap = 'gray')
plt.title('Input Image')
plt.xticks([])
plt.yticks([])

plt.subplot(122)
plt.imshow(img_new, cmap = 'gray')
plt.title('Image after filtering')
plt.xticks([])
plt.yticks([])
plt.show()
```

Результат применения фильтра Баттерворта



2.7. Частотные методы. Фильтр Винера

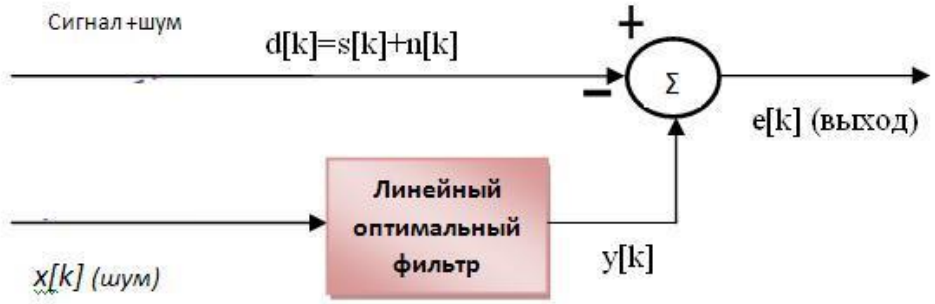
Программные средства обработки потоков (flows) информации с фотоприемников имеют ряд особенностей, что связано, прежде всего, с двумя факторами: непрерывностью обработки и с ограниченностью ресурсов. Первый фактор определяет связь времени, которое отводится на обработку, с пространственно-временной структурой сигнала. Второй фактор связан с тем, что в реальных системах ограничены возможности по производительности процессоров, параллельности обработки и объему памяти. Всегда решается вопрос об оптимальности выбора алгоритма обработки при указанных выше ограничениях.

Рассмотрим задачу фильтрации изображения от шумов, используя фильтр Винера.

Общеизвестная теория.

Фильтр Винера называется также линейным оптимальным фильтром,

поскольку меньшее значение среднеквадратической ошибки, чем в фильтре Винера, в любом линейном фильтре получить нельзя



На вход фильтр поступают два сигнала: x[k] и d[k]. При этом d[k] содержит две составляющие – полезный сигнал s[k], который не коррелирован с x[k] и шумовую составляющую n[k], коррелированную с x[k]. Фильтр Винера должен иметь такую частотную характеристику, которая обеспечивает на выходе оптимальную в среднеквадратическом смысле оценку y[k] коррелированной части сигнала (шума) n[k]. Эта оценка вычитается из d[k] и выход (ошибка) фильтра e[k] – это наилучшая по среднеквадратическому критерию оценка полезного сигнала. Таким образом, фильтр Винера обеспечивает оптимальную оценку полезного сигнала, смешанного с аддитивным шумом, по критерию минимума среднеквадратической ошибки.

Предполагается, что фильтр является КИХ-фильтром с L-го порядка (с L коэффициентами). При этом его выход вычисляется как:

$$y[k] = W^T X[k]$$

Где $W = [w_0, w_1, \dots, w_{L-1}]^T$ вектор коэффициентов фильтра;

$X[k] = [x[k], x[k-1], \dots, x[k-(L-1)]]^T$ - вектор входного сигнала

Для оптимальной фильтрации необходимо найти оптимальный вектор коэффициентов W*. Для этого необходимо найти функцию среднеквадратической ошибки, взять ее производную и приравнять к

нулю:

$e[k] = d[k] - y[k]$;

$e[k] = d[k] - W^T X[k] = d[k] - X[k]^T W$;

$e^2[k] = d^2[k] + W^T X[k] X^T[k] W - 2d[k] X^T[k] W$;

$E[e^2[k]] = E[d^2[k]] + W^T E[X[k] X^T[k]] W - 2E[d[k] X^T[k]] W$ - функция СКО.

Чтобы удобнее представить функцию СКО, необходимо ввести следующие обозначения:

$R = E[X[k] X^T[k]]$

Эта матрица называется корреляционной матрицей входного сигнала. Элементы, расположенные на главной диагонали, равны среднеквадратическим значениям входных компонентов, а остальные элементы – значениям автокорреляционной функции входных компонентов.

$P = E[d[k] X[k]]$

Этот вектор представляет собой множество значений взаимной корреляционной функции полезного отклика и отсчетов входного сигнала.

Теперь можно записать:

$E\left[e^2[k]\right] = \xi = E[d^2[k]] + W^T R W - 2P^T W$

Теперь, если взять производную от этой функции по W и приравнять ее к нулю, получим:

$W^* = R^{-1} P$

Это равенство называется уравнением Винера-Хопфа. Оно определяет вектор коэффициентов фильтра, обеспечивающий минимальное значение среднеквадратической ошибки оценки полезного сигнала в присутствии шума.

Реализация.

Было взято исходное изображение 1000x1000, затем оно было подвергнуто зашумлению и смазыванию. После, к уже испорченной картинке применялся фильтр Винера из библиотеки scipy с разным размером окна фильтрации.

Таблица использованных функций

cv2.imread	Функция imread загружает изображение из указанного файла и возвращает его. Если изображение не может быть прочитано (из-за отсутствия файла, неправильных разрешений, неподдерживаемого или недопустимого формата), функция возвращает пустую матрицу (Mat::data==NULL).
cv2.imwrite	Функция imwrite сохраняет изображение в указанный файл. Формат изображения выбирается на основе расширенияпараметра filename (список расширений см. в разделе cv::imread). В общем, только 8-битные одноканальные или 3-канальные (с порядком каналов "BGR") изображения может быть сохранены с помощью этой функции
cv2.split	Разделяет многоканальный массив на несколько одноканальных
cv2.merge	Объединяет вектор одноканальных массивов в один многоканальный массив
scipy.convolve2d	Производит свертку двух матриц в зависимости от параметра node
scipy.wiener	Применяет фильтр Винера к N-мерному массиву im: N-мерный массив. mysize: int или array_like, опционально Скалярный или N-образный список, задающий размер фильтра Винера окно в каждом измерении. Элементы mysize должны быть нечетными. Если mysize является скаляром, то этот скаляр используется в качестве размера в каждом измерении. noise : float, опционально Шум-сила шума для использования. Если нет, то шум оценивается как среднее значение локальной дисперсии входных данных.

Программа

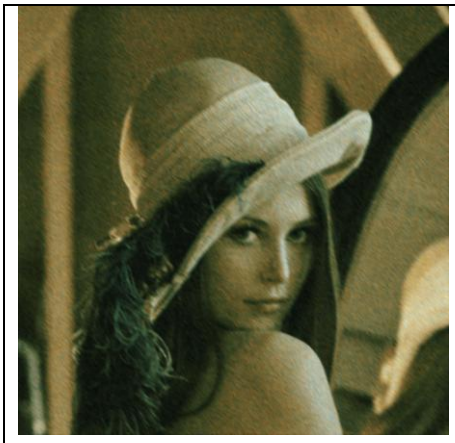
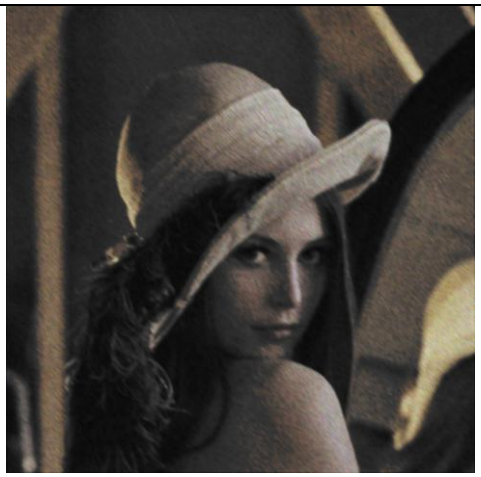
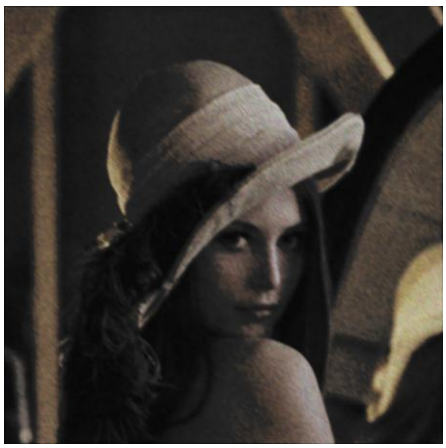
```
Зашумление.py
import cv2
import numpy as np
from scipy.signal import convolve2d

def add_noise(img, sigma):
    gauss = np.random.normal(0, sigma, np.shape(img))
    noisy_img = img + gauss
    noisy_img[noisy_img < 0] = 0
    noisy_img[noisy_img > 255] = 255
    return noisy_img

def blur(img, kernel_size):
    dummy = np.copy(img)
```

```
    h = np.eye(kernel_size) / kernel_size
    dummy_b, dummy_g, dummy_r = cv2.split(dummy)
    dummy_b = convolve2d(dummy_b, h, mode='valid')
    dummy_g = convolve2d(dummy_g, h, mode='valid')
    dummy_r = convolve2d(dummy_r, h, mode='valid')
    dummy = cv2.merge([dummy_b, dummy_g, dummy_r])
    return dummy
```

Main.py

```
import cv2
from scipy.signal.signaltools import wiener
from Зашумление import add_noise, blur

if __name__ == '__main__':
    # read image
    img_src = cv2.imread('lena1000p.jpg')
    print('IMG_size = {}x{}'.format(img_src.shape[0], img_src.shape[1]))

    # noisy_img = add_noise(img_src, 50)
    noisy_img = blur(img_src, 7)

    # save noisy image
    cv2.imwrite('noisy.jpg', noisy_img)

   # for kernel_size in range(3, 13, 2):  # размер квадратной матрицы фильтрации
        # Filter the image
       # filtered_img = wiener(noisy_img, (kernel_size, kernel_size, kernel_size))

        # save result image
       # cv2.imwrite('result' + str(kernel_size) + '.jpg', filtered_img)

    for noise in range(1, 200):
        filtered_img = wiener(noisy_img, (7, 7, 7), noise / 10)
        cv2.imwrite('result' + str(noise * 10) + '.jpg', filtered_img)
```

Исходное изображение	Зашумленное изображение
	
Результаты после фильтрации	
Окно 3x3	Окно 5x5

	
Окно 7x7 	Окно 11x11 

2.8. Сглаживание и повышение резкости

Регулировка резкости усиливает четкость краев на изображении или наоборот, сглаживает их. Регулировка резкости позволяет улучшить качество большинства изображений независимо от того, каким образом они получены. Ниже представлены примеры работы программ с применением функций OpenCV и PIL (Python Imaging Library).

OpenCV. Для повышения резкости используется фильтрация. Более высокие частоты контролируют края, а более низкие частоты контролируют содержание изображения. Края формируются, когда есть резкий переход от одного значения пикселя к другому значению пикселя, например 0 и 255 в соседней ячейке.

Для сглаживания изображения используются два метода: bilateralFilter и GaussianBlur. Двусторонний фильтр - это нелинейный сглаживающий фильтр с сохранением границ и уменьшением шума для изображений. Он заменяет интенсивность каждого пикселя средневзвешенным значением интенсивности из соседних пикселей. Гауссова фильтрация - это средневзвешенное значение интенсивности соседних позиций с весом, уменьшающимся с пространственным расстоянием до центральной позиции.

Использованные функции

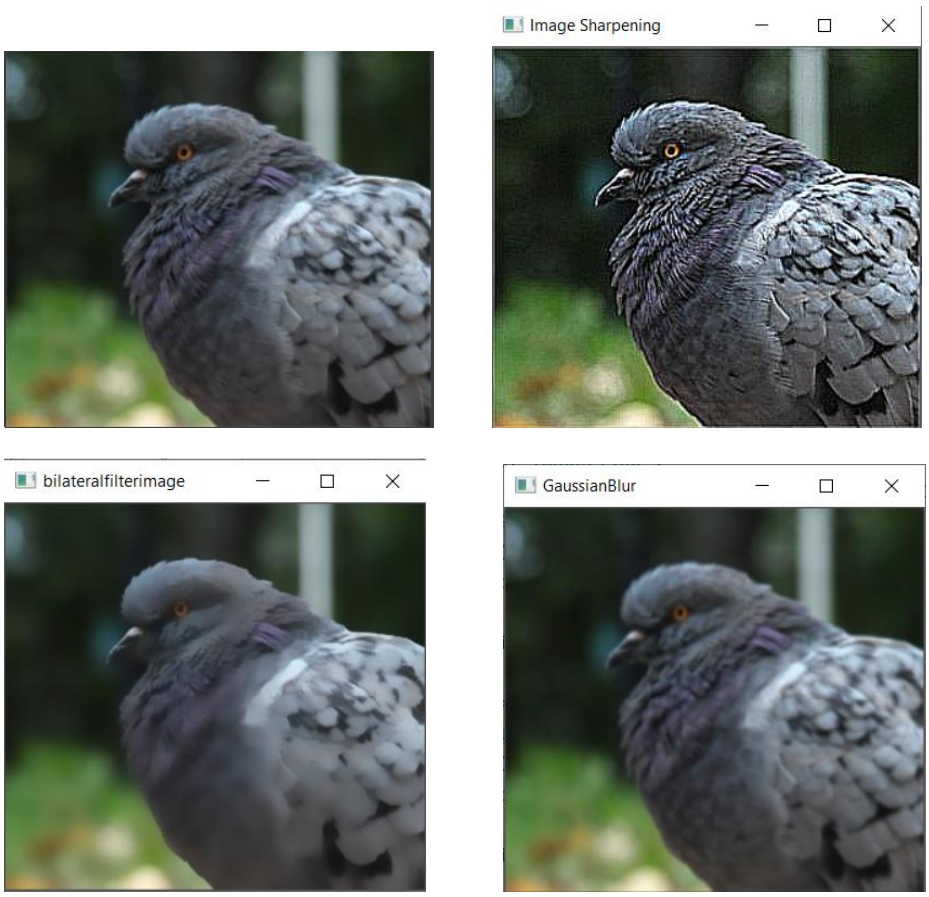
Функция	Описание
cv2.imread()	Функция, при помощи которой мы считываем изображение
cv2.filter2D()	Это метод фильтрации изображения. Свертка происходит между исходным изображением и ядром.
cv2.imshow	Используется для отображения изображения в окне.
cv2.waitKey(0)	Команда останавливает выполнение скрипта до нажатия клавиши на клавиатуре. Параметр **0** означает что нажатие любой клавиши будет засчитано.
cv2.destroyAllWindows()	Закрывает все открытые в скрипте окна.
cv2.namedWindow()	Создание окна.

Листинг программы

```
import cv2
import numpy as np
# Сглаживание и повышение резкости opencv

image = cv2.imread('image2.jpg')
kernel = np.array([[-1, -1, -1],
                   [-1, 9, -1],
                   [-1, -1, -1]])
sharpened = cv2.filter2D(image, -1, kernel)
cv2.imshow('Image Sharpening', sharpened)
cv2.waitKey(0)
cv2.destroyAllWindows()

img = cv2.imread("image2.jpg")
gaussian_blur = cv2.GaussianBlur(img, (5, 5), sigmaX=0)
window_name = 'GaussianBlur'
cv2.namedWindow(window_name, cv2.WINDOW_NORMAL)
cv2.imshow(window_name, gaussian_blur)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()

bilateral_blur = cv2.bilateralFilter(img,15,80,80)
window_name='bilateralfilterimage'
cv2.namedWindow(window_name, cv2.WINDOW_NORMAL)
cv2.imshow(window_name,bilateral_blur)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Результат работы программы OpenCV исходная картинка, повышение резкости, сглаживание: bilateralFilter, GaussianBlur



bilateralFilter *GaussianBlur*

PIL (Python Imaging Library)—это бесплатная Python-библиотека для открытия, работы и сохранения различных форматов изображений.

Использованные функции

Функция	Описание
Image.open()	Возвращает объект типа класса PIL.PngImagePlugin.PngImageFile. Иными словами Загружает изображение из указанного файла

ImageEnhance.Sharpness()	Этот класс можно использовать для регулировки резкости изображения
Enhance()	усилитель

Листинг программы

```
from PIL import Image, ImageEnhance
factor = 4
im_s_1 = enhancer.enhance(factor)
im_s_1.save('sharpened-image.jpg');

im = Image.open("sharpened-image.jpg")
enhancer = ImageEnhance.Sharpness(im)

factor = 0.0005
im_s_1 = enhancer.enhance(factor)
im_s_1.save('blurred-image.jpg')
```

Результат работы программы PIL

Повышенная резкость изображения

Сглаженное изображение

2.9. Подмена пикселей

Пиксельные преобразования – это те преобразования, где входом является пиксель (несколько близлежащих пикселей), и соотвествующим выходом также является пиксель. Это самый простой вид операций, применяемых к изображению. Такие преобразования могут быть смоделированы как функции, которые принимают одно или несколько входных изображений и создают выходное изображение. Также их называют точечными операторами (Point Operator). В непрерывной области их можно обозначить как:

$$g(x) = h\big(f_0(x), f_1(x), \dots, f_n(x)\big),$$

где g – получаемое изображение, x – массив размера N, f_0 – первый точечный оператор.

Примером такого пиксельного преобразования является регулировка контрастности. Можно отрегулировать контраст и яркость изображения, применяя локальные точечные операторы. Умножение отдельных пикселей изображения на константу и добавление смещения приведет к тому, что изображение станет более или менее ярким, а контраст более или менее заметным.

В OpenCV имеется функция convertScaleAbs. Она на каждом элементе входного массива последовательно выполняет три операции: масштабирование, получение абсолютного значения, преобразование в 8-битный тип без знака.

Формат функции:

`void convertScaleAbs (SRC, dst, альфа = 1, бета =0 )`

`src` - входной массив.

`dst` - выходной массив.

`альфа` - необязательный масштабный коэффициент.

`бета` - необязательная дельта добавляется к масштабированным значениям.

Если выходной сигнал не 8-битный, операцию можно эмулировать, вызвав метод `Mat:: convertTo` (или используя матричные выражения).

```
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

int main ( int argc, char** argv  ) {

    Mat image;
    int x=0, y=0;
    image = cv::imread("picture.jpg", IMREAD_COLOR );
    contrast = 3.0
    brightness = 100
```

```
    Mat result = convertScaleAbs(image, result, contrast, brightness)
    cv.imshow('Resulting Image', result)
    waitKey(0);
    return 0;
```

Алгоритм подмены пикселей сводится к операции взятия пикселей из одной картинки/видео и запись их в другую картинку/видео. Такой алгоритм позволяет, например, в текущее видео вставлять другое видео или картинку. Особенностями такой операции являются функции определениея граничных точек вставляемого изображения, изменения размера изображения. Для изменения размера изображения имеется функция библиотеки OpenCV `cv2.resize ().`

Ниже приведена программа, в которой одно видео `'v4.mp4'` вставлено в другое `'v3.mp4'`. Размер первого видео уменьшается на четверть.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import matplotlib
matplotlib.use('Qt5Agg')
import numpy
from scipy.ndimage import label

cap1 = cv2.VideoCapture('v3.mp4')
cap2 = cv2.VideoCapture('v4.mp4')

while cap1.isOpened():
    ret1, frame1 = cap1.read()
    ret2, img = cap2.read()
    img = cv2.resize(img, (0, 0), fx=0.25, fy=0.25)

    x_offset = 300
    y_offset = 20
    frame1[y_offset:y_offset+img.shape[0], x_offset:x_offset+img.shape[1]] = img

    # render
    cv2.imshow('ORIGINAL', frame1)  # original
    cv2.moveWindow("ORIGINAL", 0, 0)

    # time.sleep(0.05)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap1.release()
cv2.destroyAllWindows()
```

2.10. Изменение палитры цветов, псевдо раскраска

Различные фотоприемники или камеры выдают различные значения RGB для одного и того же цвета. Настройка палитры цветов на данной стадии заключается в создании матрицы смешивания, позволяющей преобразовать одно цветовое пространство фотоприемника в другое цветовое пространство. В процессе преобразования возможно или только изменить кодирование пикселей, или изменить их цветовые представления.

Известны стандартные палитры цветов: REC.709 (sRGB), Adobe RGB, DCI-P3, BT.2020

REC.709 (sRGB) представляет собой стандарт для телевидения высокой четкости. Он определяет кодирование R'G'B' и кодирование $Y'C_B C_R$, каждое с 8 битами или 10 битами на выборку в каждом цветовом канале. Основные характеристики:

Разрешение 1280 x 720 (HD) или 1920 x 1080 (FHD) пикселей

8/10-битная глубина цвета на каждый канал

Цветовая субдискретизация 4:2:0

Прогрессивная развертка до 60 кадров в секунду

Цветовой охват Rec. 709 (35,9% от пространства CIE 1931)

DCI-P3 или DCI/P3 (DCI - Digital Cinema Initiatives) - цветовое пространство, используемое в цифровых кинотеатрах. Многие продукты от Apple, Sony, Samsung и Google передают цвет гораздо лучше, чем их предшественники именно благодаря использованию DCI-P3. В ближайшем будущем этот стандарт будет заменять sRGB в качестве нового эталона для цифровых устройств, веб-сайтов и приложений.

BT.2020 или Rec.2020 – стандарт для Ultra HD. Rec.2020 имеет следующие характеристики изображения:

Разрешение 3840 x 2160 (4K) или 7680 x 4320 (8K) пикселей

10/12-битная глубина цвета на каждый канал

Цветовая субдискретизация 4:2:0, 4:2:2 или 4:4:4

Прогрессивная развертка 60—120 кадров в секунду

Широкий цветовой охват Rec. 2020 (75,8% от пространства CIE 1931)

Для преобразования изображения из одного цветового пространства в другое в OpenCV используется функция cvtColor().

```
cvtColor(src, bwsrc, cv::COLOR_RGB2GRAY)
```

Функция позволяет проводить преобразования в пространстве RGB, такие как добавление / удаление альфа-канала, изменение порядка каналов, преобразование в/из 16-битного цвета RGB (R5: G6: B5 или R5: G5: B5), а также преобразование в / из оттенков серого с использованием:

- RGB [A] в серый: Y ← 0,299 0.2R + 0,587·G + 0,114·B
- От серого до RGB [A]: R ← Y, G ← Y, B ← Y, A ← max (ChannelRange)

Конвертация из RGB в Rec 709 происходит по следующему алгоритму:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019335 & 0.119193 & 0.950222 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} \leftarrow \begin{bmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Конвертация из RGB в YCrCb происходит по следующему алгоритму:

$$Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$
$$Cr \leftarrow (R - Y) \cdot 0.713 + delta$$
$$Cb \leftarrow (B - Y) \cdot 0.564 + delta$$
$$R \leftarrow Y + 1.403 \cdot (Cr - delta)$$
$$G \leftarrow Y - 0.714 \cdot (Cr - delta) - 0.344 \cdot (Cb - delta)$$
$$B \leftarrow Y + 1.773 \cdot (Cb - delta),$$

где:

$$delta = \begin{cases} 128 \text{ для 8 бит} \\ 32768 \text{ для 16 бит} \\ 0.5 \text{ для плавающей точки} \end{cases}$$

Конвертация из RGB в CIE L*a*b* в случае 8-битных и 16-битных изображений R, G и B преобразуются в формат с плавающей запятой и масштабируются, чтобы соответствовать диапазону от 0 до 1 и происходит по следующему алгоритму:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019335 & 0.119193 & 0.950222 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$X \leftarrow X/X_n, \text{ где } X_n = 0.950456$$
$$Z \leftarrow Z/Z_n, \text{ где } Z_n = 1.088754$$

$$L \begin{cases} 116 * Y^{\frac{1}{3}} - 16 \; Y > 0.008856 \\ 903.3 * Y \leq 0.008856 \end{cases}$$

$$a \leftarrow 500 \; (f(X) - f(Y)) + delta$$
$$b \leftarrow 200 (f(Y) - f(Z)) + delta$$

где:

$$f(t) = \begin{cases} t^{\frac{1}{3}}, \text{для } t > 0.008856 \\ 7.787t + 16/116, \text{для } t \leq 0.008856 \end{cases}$$

и где:

$$delta = \begin{cases} 128 \text{ для 8 битных изображений} \\ 0 \text{ для плавающей точк} \end{cases}$$

Конвертация с использованием представленных формул не представляет сложностей.

Подстройка палитры видео к палитре представленного изображения

Задача ставится следующим образом: по образцу цветовой палитры статического изображения перекрашивается видео. Приложение должно позволять загрузить изображение, разделить на 3 канала (RGB), определить доминирующие цвета, перенести цвета на видео и вывести результат.

Этапы определения доминирующих цветов и переноса на видео являются эвристической задачей, которую можно решить по-разному. Ниже представлен самой простой алгоритм.

```
// подключение необходимых библиотек
#include <vector>

#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

// Макрос для имени окна
#define MODIFIED_NAME "Modified image"

int main() {
    // считывание изображения
    cv::Mat mask = cv::imread("/home/fedor/CLionProjects/lab7/1.jpg",
cv::IMREAD_COLOR);
    // изменение размера изображения
    cv::resize(mask, mask, cv::Size(300, 200));
    // конвертирование в 32FC3
    mask.convertTo(mask, CV_32FC3, 1/255.0);
    // вектор для хранения каналов
    std::vector<cv::Mat> mask_channels;
    // вектор для хранения каналов
    cv::split(mask, mask_channels);

    // нормализация изображения
    cv::normalize(mask, mask, 0, 255, cv::NORM_MINMAX, CV_32FC3);

    // объект для работы с видео
    cv::VideoCapture cap("/home/fedor/CLionProjects/lab7/1.mp4");

    static cv::Mat image;

    while (cap.isOpened()) {
    // считывание видео в цикле
        cap >> image;
        // изменение размер
        resize(image, image, cv::Size(300, 200));
        // конвертация в 32FC3
        image.convertTo(image, CV_32FC3, 1/255.0);

    // вектор для каналов
        std::vector<cv::Mat> im_channels;
        // разделение кадра видео на каналы
        cv::split(image, im_channels);
        // нормализация
        cv::normalize(image, image, 0, 255, cv::NORM_MINMAX, CV_32FC3);

        // умножение нормализованных пикселей изображения и кадра видео
        for (auto i = 0; i < im_channels.size(); ++i) {
            im_channels[i] = im_channels[i].mul(mask_channels.at(i));
        }

        cv::Mat plot;
```

```
            // передача значений каналов в объект cv::Mat
            cv::merge(im_channels, plot);

            // конвертация изображения в CV_8UC3
            image.convertTo(image, CV_8UC3);
            // вывод изображени
            cv::imshow(MODIFIED_NAME, plot);
        }

        return 0;
```

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import cv2
from collections import Counter

def get_colors(number_of_colors, show_chart):

    image = cv2.imread('l18.jpg')
    print("The type of this input is {}".format(type(image)))
    print("Shape: {}".format(image.shape))
    plt.imshow(image)

    ## Output
    # The type of this input is <class 'numpy.ndarray'>
    # Shape: (3456, 4608, 3)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.imshow(image)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    plt.imshow(gray_image, cmap='gray')
    resized_image = cv2.resize(image, (1200, 600))
    plt.imshow(resized_image)

    def RGB2HEX(color):
        return "#{:02x}{:02x}{:02x}".format(int(color[0]), int(color[1]), int(color[2]))

    def get_image(image_path):
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        return image

    modified_image = cv2.resize(image, (600, 400), interpolation = cv2.INTER_AREA)
    modified_image = modified_image.reshape(modified_image.shape[0]*modified_image.shape[1], 3)

    clf = KMeans(n_clusters = number_of_colors)
    labels = clf.fit_predict(modified_image)

    counts = Counter(labels)
    center_colors = clf.cluster_centers_
    # We get ordered colors by iterating through the keys
    ordered_colors = [center_colors[i] for i in counts.keys()]
    hex_colors = [RGB2HEX(ordered_colors[i]) for i in counts.keys()]
    rgb_colors = [ordered_colors[i] for i in counts.keys()]

    if (show_chart):
        plt.figure(figsize=(8, 6))
        plt.pie(counts.values(), labels=hex_colors, colors=hex_colors)
        plt.waitforbuttonpress()
    return rgb_colors

get_colors(6, True)
```

2.11. Определение HSV-кода пикселя в цветовом цилиндре

Программа должна вывести значения HSV-кода (H - тон, S - насыщенность, V – значение (яркость)) пикселя, на который пользователь указывает по картинке цветового цилиндра. Цветовой цилиндр выводится программой.

Диапазоны значений:

H – (0; 360)
S – (0; 1)
V – (0; 255)
Передлагаемая программа выполняет следующие действия:

- Получает размер картинки
- Запоминает в массиве HSV кодировку всех пикселей
- Показывает картинку пользователю
- После левого клика мышкой, программа получает координаты пикселя
- Программа рядом с нажатым пикселем выводит HSV кодировку данного пикселя, также можно нажать Enter чтобы программа вывела HSV-код в консоли.

Суть сложности в том, что при получении цвета через координаты пикселя можно получить только RGB кодировку, но можно использовать данную формулу, чтобы получить HSV код:

Пусть MAX — максимальное значение из R, G и B, а MIN — минимальное из них.

$$H = \begin{cases} 0, \text{ если } MAX = MIN \\ 60 \times \frac{G - B}{MAX - MIN} + 0, \text{ если } MAX = R \text{ и } G \geq B \\ 60 \times \frac{G - B}{MAX - MIN} + 360, \text{ если } MAX = R \text{ и } G < B \\ 60 \times \frac{B - R}{MAX - MIN} + 120, \text{ если } MAX = G \\ 60 \times \frac{R - G}{MAX - MIN} + 240, \text{ если } MAX = B \end{cases}$$

$$S = \begin{cases} 0, \text{ если } MAX = 0; \\ 1 - \frac{MIN}{MAX}, \text{ иначе} \end{cases}$$

$$V = MAX$$

Использованные функции

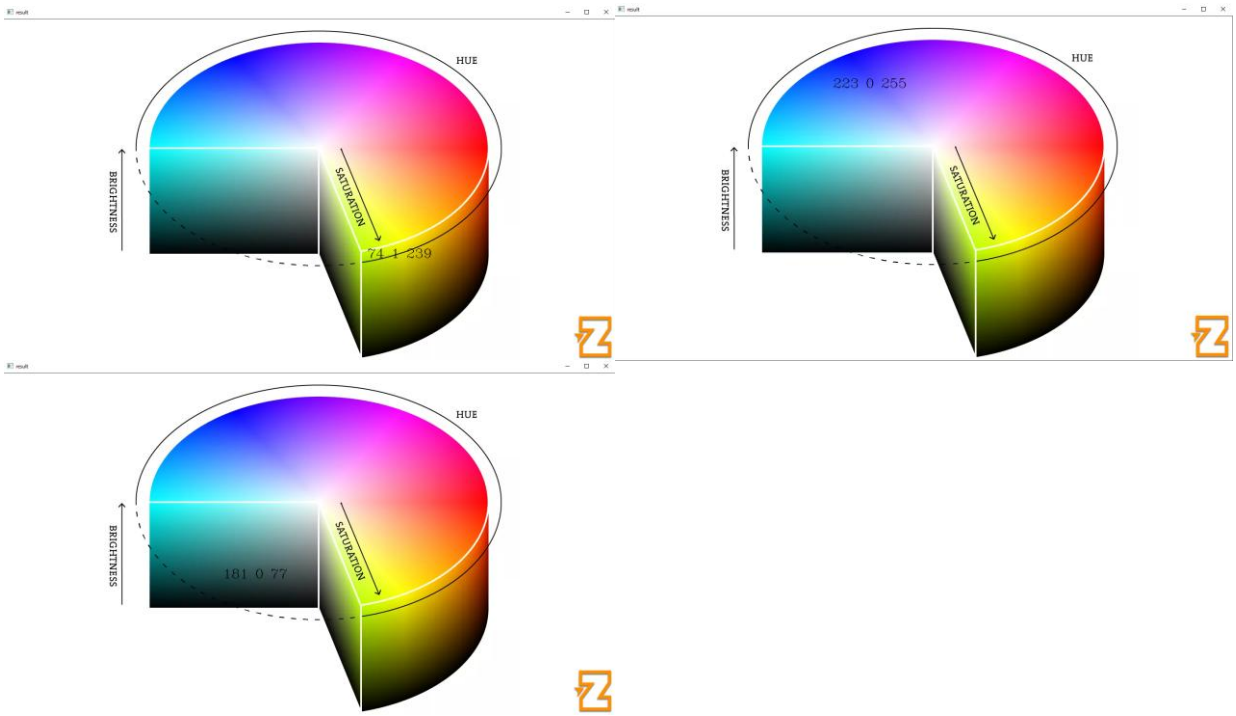
Функция	Описание
setMouseCallback('rez', draw_circle)	Ловит события от мышки по окну, имя которого rez и вызывает функцию getXY.
putText(rez, string, (x, y), font, 1, (0,0,0))	Вывести текст string в окне rez, по координатам x, y.
waitKey(5)	Ожидание получения кода нажатой кнопки на клавиатуре
destroyAllWindows()	Закрывает все окна, которые мы открыли
imread(file)	Прочитывает изображения по местоположению file

Программа

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 10 17:39:13 2021
@author: Сашка
"""

"""
Created on Thu Apr  8 20:47:22 2021
@author: Сашка
"""

import cv2
import numpy as np
if __name__ == '__main__':
    def nothing(*arg):
        pass

#Получение координат мышки
def getXY(event,x,y,flags,param):
    global mouseX, mouseY
    if event == cv2.EVENT_LBUTTONDBLCLK:
        mouseX = x
        mouseY = y

#Прочитываем изображение
img = cv2.imread('D:\HSV\HSV.png', cv2.IMREAD_COLOR)

#Находим высоту и ширину изображения
height = np.size(img,0)
width = np.size(img,1)

hsv = np.arange(height*width*3).reshape(height,width,3)

font = cv2.FONT_HERSHEY_COMPLEX

mouseX = 0
mouseY = 0

#Запоминаем цвета пикселей
for i in range(0,width):
    for j in range(0,height):
```

```
        R = int(img[j,i][2])
        G = int(img[j,i][1])
        B = int(img[j,i][0])

        maximum = max(R,G,B)
        minimum = min(R,G,B)

        #Перевод из BGR в HSV
        #hsv[j][i][0] - HUE; hsv[j][i][1] - SATURATION; hsv[j][i][2] - VALUE
        if (maximum == minimum):
            hsv[j][i][0] = 0
        elif (maximum == R) :
            if(G >= B) :
                hsv[j][i][0] = 60 * (G-B)/(maximum-minimum)
            else:
                hsv[j][i][0] = 60 * (G-B)/(maximum-minimum) + 360
        elif (maximum == G) :
            hsv[j][i][0] = 60 * (B-R)/(maximum-minimum) + 120
        elif (maximum == B) :
            hsv[j][i][0] = 60 * (R-G)/(maximum-minimum) + 240

        if(maximum == 0) :
            hsv[j][i][1] = 0
        else :
            hsv[j][i][1] = 1 - (minimum/maximum)

        hsv[j][i][2] = maximum

while True:
    #Прочитываем изображение
    image = cv2.imread('D:\HSV\HSV.png')

    #Ловит событие от мышки
    cv2.setMouseCallback('result',getXY)

    #Вывод текста на изображении
    if(mouseX!=0 and mouseY!=0) :
        string = str(hsv[mouseY][mouseX][0]) + " " + str(hsv[mouseY][mouseX][1]) +
" " + str(hsv[mouseY][mouseX][2])
        cv2.putText(image, string, (mouseX+3,mouseY+3), font, 1, (0, 0, 0))

    #Показывает изображение
    cv2.imshow('result', image)
    ch = cv2.waitKey(5)
    if ch == 27:
        break
    elif ch == 13:
         print(" HUE: ",hsv[mouseY][mouseX][0]," SATURATION: ",
hsv[mouseY][mouseX][1]," VALUE: ", hsv[mouseY][mouseX][2])

#Закрывает все окна
cv2.destroyAllWindows()
```

Примеры работы программы



2.12. Определение преобладающих цветов на изображении с использованием кластеризации методом k-средних

Определение преобладающих цветов на изображении представляется важной задачей при работе с палитрой цветов. Представим программу, которая выводит изображение и линейку цветов в соотношении друг с другом. Программа выполняет следующие действия :

1. Загружает изображение из файла
2. Создает объект, который преобразован из картинки изображения в список пикселей этой картинки.
3. Находим кластер по методу к-средних
4. Визуализируем полученный кластер

При расчете преобладающего цвета используется кластеризация методом k-средних. Кластеризация - это задача группировка набора объектов таким образом, чтобы объекты в одной группе (называемой кластером) были более похожи (в некотором смысле) друг на друга, чем на объекты в других группах (кластерах). Для к-средних кластеры представлены центральным вектором, когда количество кластеров

фиксировано на k. найти k центров кластеров и назначить объекты ближайшему центру кластера, так чтобы квадраты расстояний от кластера были минимизированы.

Использованные функции

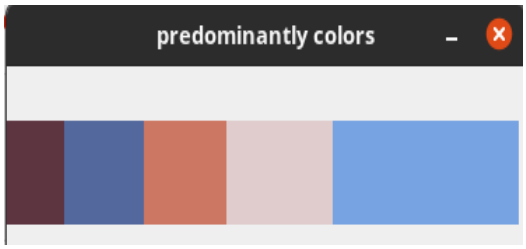
Функция	Описание
KMeans(n_clusters=5).fit(reshape)	Используемый алгоритм K-средних.
np.arange(0, len(np.unique(cluster.labels_)) + 1)	Получение уникального количества кластеров
np.histogram(cluster.labels_, bins = labels)	Создание и нормализация гистограммы
cv2.rectangle(rect, (int(start), 0), (int(end), 50), color.astype("uint8").tolist(), -1)	Создание визуальной прямоугольной гистограммы

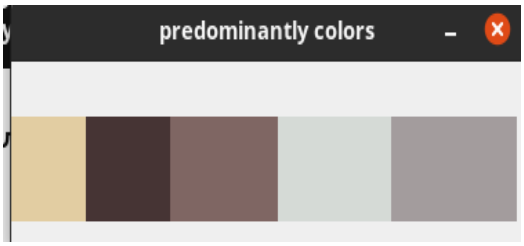
Программа

```
import cv2, numpy as np
from sklearn.cluster import KMeans

def visualize_colors(cluster, centroids):
   # получение количества различных кластеров
   # создание и нормализация гистограммы
   labels = np.arange(0, len(np.unique(cluster.labels_)) + 1)
   (hist, _) = np.histogram(cluster.labels_, bins = labels)
   hist = hist.astype("float")
   hist /= hist.sum()

   # Создаем прямоугольник частоты и перебираем
   # цвет и процент каждого кластера
   rect = np.zeros((50, 300, 3), dtype=np.uint8)
   colors = sorted([(percent, color) for (percent, color) in zip(hist,
centroids)])
   start = 0
   for (percent, color) in colors:
       print(color, "{:0.2f}%".format(percent * 100))
       end = start + (percent * 300)
       cv2.rectangle(rect, (int(start), 0), (int(end), 50),
color.astype("uint8").tolist(), -1)
       start = end
   return rect

# Загрузка изображения и преобразование в список пикселей
image = cv2.imread('photos/ph3.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
reshape = image.reshape((image.shape[0] * image.shape[1], 3))

# Нахождение и отображение наиболее доминирующего цвета
cluster = KMeans(n_clusters=5).fit(reshape)
visualize = visualize_colors(cluster, cluster.cluster_centers_)

visualize = cv2.cvtColor(visualize, cv2.COLOR_RGB2BGR)
```

```
cv2.imshow('visualize', visualize)
cv2.waitKey()
```

Примеры применения программы





2.13. Определение преобладающих цветов на изображении с использованием HSV палитры

Программа находит на изображении указанное количество преобладающих цветов и демонстрирует их. Программа выполняет следующие действия:

1. Загружает изображение из файла
2. Переводит изображение в HSV формат
3. Определяет преобладающие цвета
4. Переводит обратно в RGB формат и выводит преобладающие цвета

При расчете преобладающих цветов используется HSV формат (H - тон, S - насыщенность, V - значение (яркость))

Диапазоны значений: H (0-180), S (0-255), V (0-255)

Преобладание цвета определяются по компоненте H. Подсчитывается количество вхождений каждого тона, и средние насыщенность и яркость для каждого тона. Затем они объединяются в участки по 5 штук, чтобы очень похожие цвета считались за один. Среди этих участков находится тот, количество вхождений которого наибольшее, он и считается преобладающим. Далее может искаться преобладающий среди оставшихся.

Использованные функции

Функция	Описание
cv2.imread('pict.bmp')	Загрузка изображения
cv2.cvtColor(image, cv2.COLOR_BGR2HSV)	Перевод изображения из BGR формата в HSV формат
cv2.cvtColor(image, cv2.COLOR_HSV2BGR)	Перевод изображения из HSV формата в BGR формат
cv2.resize(col, (800, 800))	Масштабирование изображения к размеру 800 на 800
cv2.imshow("Original image", image)	Отрисовка изображения в окне с именем "Original image"
cv2.waitKey(0)	Ожидание нажатия клавиши

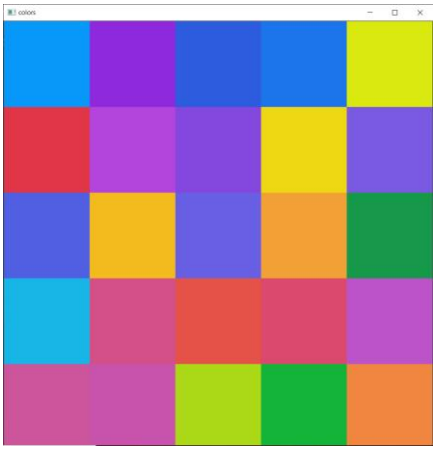
Листинг программы

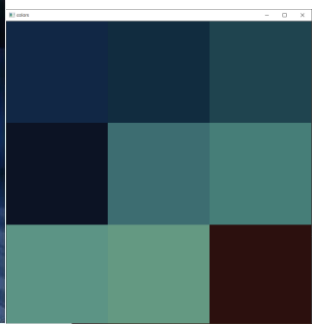
```
import cv2
import math

# Заполняет квадрат с номером imageIndex картинки image переданным цветом.
def fillColorInHSV(image, hue, saturation, value, imageNumber, imageIndex):
    size = math.ceil(math.sqrt(imageNumber))
    xscaleCoeff = image.shape[0] / size
    yscaleCoeff = image.shape[1] / size
    widthStartIndex = math.floor((imageIndex % size) * xscaleCoeff)
    widthEndIndex = math.floor((imageIndex % size + 1) * xscaleCoeff)
    heightStartIndex = math.floor(math.floor(imageIndex / size) * yscaleCoeff)
    heightEndIndex = math.floor(math.floor(imageIndex / size + 1) * yscaleCoeff)
    for i in range(heightStartIndex, heightEndIndex):
        for j in range(widthStartIndex, widthEndIndex):
```

```
            image[i][j] = [hue, saturation, value];

# Вырисовывает картинку с num преобладающих цветом
def showDominateColors(image, num):
    # Перевод картинки из RGB формата в HSV формат
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    counts = []
    saturation = []
    value = []
    # Заполнение массивов нулями
    for i in range(180):
        counts.append(0)
        saturation.append(0)
        value.append(0)

    # Подсчет пикселей с каждым тоном и вычисление
    # для каждого тона средние насыщенность и яркость
    for i in range(hsv.shape[0]):
        for j in range(hsv.shape[1]):
            if hsv[i][j][1] != 0:
                counts[hsv[i][j][0]] += 1
                saturation[hsv[i][j][0]] += hsv[i][j][1]
                value[hsv[i][j][0]] += hsv[i][j][2]
    for i in range(180):
        if counts[i] != 0:
            saturation[i] /= counts[i]
            value[i] /= counts[i]

    # Объединение тонов в слои по 5 штук
    truncateCount = []
    truncateSaturation = []
    truncateValue = []
    for i in range(36):
        truncateCount.append(counts[i*5] + counts[i*5+1] + counts[i*5+2] +
                             counts[i*5+3] + counts[i*5+4])
        truncateSaturation.append((saturation[i*5] + saturation[i*5+1] +
                                   saturation[i*5+2] +
                                   saturation[i*5+3] + saturation[i*5+4]) / 5)
        truncateValue.append((value[i * 5] + value[i * 5 + 1] + value[i * 5 + 2] +
                              value[i * 5 + 3] + value[i * 5 + 4]) / 5)

    # Создание черной картинки, на которую будут заноситься преобладающие цвета
    col = hsv
    for i in range(col.shape[0]):
        for j in range(col.shape[1]):
            for k in range(3):
                col[i][j][k] = 0
    col = cv2.resize(col, (800, 800))

    # Нахождение тона с максимальным количество вхождений, добавление его
    # на картинку и исключение из массива, чтобы искать следующий
    for i in range(num):
        max = 0
        maxi = 0;
        for j in range(36):
            if truncateCount[j] > max:
                max = truncateCount[j]
                maxi = j
        if max != 0:
```

```
            fillColorInHSV(col, maxi * 5 + 2,
                           truncateSaturation[maxi], truncateValue[maxi], num, i)
        truncateCount[maxi] = 0;

    # Перевод картинки из HSV формата в RGB и ее отрисовка
    col = cv2.cvtColor(col, cv2.COLOR_HSV2BGR)
    cv2.imshow("colors", col)

if __name__ == '__main__':
    # Загрузка изображения и его отрисовка
    image = cv2.imread('pict.bmp')
    cv2.imshow("Original image", image)

    showDominateColors(image, 23)

    cv2.waitKey(0)
```

Примеры применения программы





2.14. Добавление водяного знака на видео

Водяной знак — это знак на изображении, который указывает на авторство или владельца этого изображения. Как правило, он должен быть

полупрозрачным и еле заметным. Водяной знак защищает контент от воровства и/или делает рекламу автору.

Добавление водяного знака происходит с помощью функции `addWeighted()`. Эта функция позволяет складывать изображения одного размера с заданными весами, определяющими прозрачность каждого изображения. Формат функции OpenCV:

addWeighted `(src1, alpha, src2, beta, γ, dst)`

$dst=\alpha \cdot src1+\beta \cdot src2+\gamma$

Изображение, из которого необходимо сделать водяной знак, представляет из себя картинку с расширением png. Этот формат позволяет при кодировании цвета пикселя сохранять четыре байта: красный, зелёный, синий и альфа (RGBA). Альфа-канал используется для создания эффекта частичной прозрачности в изображении.

Ниже приводится программа добавления водяного знака на видео. Она выполняет следующие действия:

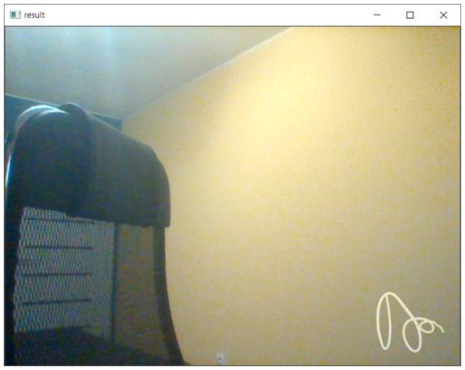
• Определение размера кадра (H, W);

• Создание пустого изображения и добавление на него водяного знака путем замены части пустого изображения;

• Добавление в кадр альфа-канала. После того как размерности изображения с водяным знаком и кадра видео стали одинаковыми, можно применять cv2.addWeighted с желаемыми весами. Для кадра оставлен вес = 1. Это означает, что кадр не будет прозрачным. Для водяного знака выбран вес = 0.3, что означает, что водяной знак будет на 70% прозрачным.

```
import cv2
import numpy as np

# Читаем первый кадр видео и определяем его высоту и ширину
capture = cv2.VideoCapture(0)
(grabbed, frame) = capture.read()
oH, oW = frame.shape[:2]
```

```
# Читаем изображение для водяного знака и инвертируем
lgo_img = cv2.imread('sign1.png', cv2.IMREAD_UNCHANGED)
lgo_img = cv2.bitwise_not(lgo_img)

scl = 50
w = int(lgo_img.shape[1] * scl / 100)
h = int(lgo_img.shape[0] * scl / 100)
dim = (w, h)
lgo = cv2.resize(lgo_img, dim, interpolation=cv2.INTER_AREA)

# Создаем полотно такого же размера как кадр видео
lH, lW = lgo.shape[:2]

# Заменяем часть полотна изображением водяного знака
ovr = np.zeros((oH, oW, 4), dtype="uint8")
ovr[oH - lH - 10:oH - 10, oW - lW - 10:oW - 10] = lgo

while True:
# Читаем кадр
    (grabbed, frame) = capture.read()

# Добавляем для кадра альфа-канал
    frame = np.dstack([frame, np.ones((oH, oW), dtype="uint8") * 255])

# Накладываем на кадр водяной знак
    final = frame.copy()
    final = cv2.addWeighted(final, 1.0, ovr, 0.3, 0, final)
    cv2.imshow('result', final)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

capture.release()
cv2.destroyAllWindows()
```



2.15. Алгоритм Retinex

Теория ретинекса была впервые предложена Лэндом и Макканном для описания модели того, как глаз воспринимает интенсивность света, которая часто расходится с фактической физической интенсивностью, которую испытывает глаз. Эта теория была значительно расширена для использования в обработке изображений с момента ее предложения.

В теории ретинекса производятся очень естественные на вид результаты, которые хорошо поддаются реализации в реальном времени на графическом процессоре.

Постоянство цвета - желательная характеристика компьютерного зрения, и для этой цели было разработано множество алгоритмов. К ним относятся несколько алгоритмов ретинекса.

Описание используемого алгоритма

Исходное изображение вместе с параметрами из файла конфигурации отправляется в функцию для обработки алгоритмом, т.к. обе сходны, то рассмотрим первую: auto_multi_scale_retinex(). Здесь преобразование в int64, представляющее целые числа в диапазоне от -9223372036854775808 по 9223372036854775807, (числа размером 8 байт) для получения численной границы. Далее применяется multiScaleRetinex() – создается множественная шкала на основе представленных в параметре sigma_list. Далее – цикл по диапазону размерности исходника. Находятся уникальные элементы и подсчитываются вместе с количеством вхождений. Прохождение проверки на наличие уникальных элементов. Далее в относительной величине измеряется относительную частоту вхождения элемента. Опять цикл со схожим назначением. И возврат изображения с измененными характеристиками.

Таблица используемых функций OpenCV

cv2.imread()	загружает изображение из указанного файла. Если изображение не может быть прочитано (из-за отсутствия файла, неправильных разрешений, неподдерживаемого или недопустимого формата), этот метод возвращает пустую матрицу.
cv2.imshow()	используется для отображения изображения в окне. Окно автоматически соответствует размеру изображения
cv2.Gaussianblur()	позволяет размывать изображения, полезно при обработке изображений. Принимает два основных параметра. Первым параметром будет изображение, а вторым - размер ядра. Модуль OpenCV Python использует ядро для размытия изображения. И ядро сообщает, насколько необходимо изменить значение данного пикселя, чтобы изображение было размыто.

auto_multi_scale_retinex()	реализует алгоритм ретинекса. Из параметров – исходное изображение (после применения cv2.imread()) и аргументы из списка sigma_list.
multi_scale_retinex()	реализует алгоритм ретинекса, с учетом интенсивности. Принимает как аргументы исходное изображение (после применения cv2.imread()) и все параметры, описанные в файле конфигурации.

Таблица используемых функций numpy

log10()	вычисляет десятичный логарифм элементов массива. Возвращает массив вычисленных значений десятичного логарифма для всех элементов входного массива или число, если на вход подано одно число.
zeros_like()	возвращает новый массив из нулей с формой и типом данных указанного массива
unique()	находит уникальные элементы массива и возвращает их в отсортированном массиве. Если return_counts – True, то помимо самих уникальных элементов так же будет возвращаться количество вхождений каждого из них во входном массиве. По умолчанию return_counts = False.
maximum()	возвращает наибольшие значения поэлементного сравнения значений массивов.
minimum()	возвращает наименьшие значения поэлементного сравнения значений массивов.
sum()	выполняет суммирование элементов массива, которое так же может выполняться по указанной оси (осям).
expand_dims()	добавляет новую ось к массиву. Длинна новой оси всегда равна 1.

run.py

```
import os
import cv2
import json
import retinex

path = 'data'
image_list = os.listdir(path)
if len(image_list) == 0:
    print("Data error")
    exit()
# output the names of files and directories contained
# in the passed parameter.
print(image_list)

# ensures that the file is closed in any case.
# open for reading with the context manager.
with open('config.json', 'r') as file_con:
    config = json.load(file_con)

for image_name in image_list:
    # method loads an image from the specified file,
    # which is formed by path and component concatenation
    img = cv2.imread(os.path.join(path, image_name))

    img_amsr = retinex.auto_multi_scale_retinex(img, config['sigma_list'])
    img_msr = retinex.multi_scale_retinex(img, config['sigma_list'],
config['low'], config['high'])
```

```
    cv2.imshow('Image', img)
    cv2.imshow('Retinex', img_amsr)
    cv2.imshow('multi_scale_retinex', img_msr)
    cv2.waitKey()
```

retinex.py

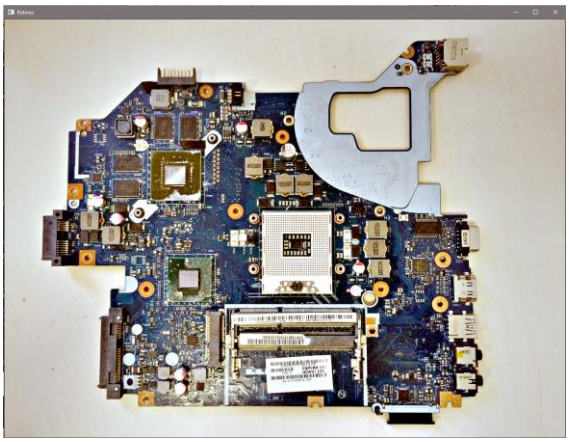
```
import numpy as np
import cv2
def singleScaleRetinex(img, sigma):
    """
    Create a single scale by comparing two logarithmic scales.
    :param img: source
    :param sigma: degree
    :return: operation result
    """
    retinex = np.log10(img) - np.log10(cv2.GaussianBlur(img, (0, 0), sigma))
    return retinex
def multiScaleRetinex(img, sigma_list):
    """
    Multiple scale is created by considering a single scale evaluation function
for each image from the image list.
    :param img: source
    :param sigma_list: list sigma (degrees)
    :return: operation result (average value)
    """
    retinex = np.zeros_like(img)
    for sigma in sigma_list:
        retinex += singleScaleRetinex(img, sigma)
    retinex = retinex / len(sigma_list)
    return retinex
def simplestColorBalance(img, low, high):
    """
    Creates the simplest color balance based on the received parameters.
    :param img: source
    :param low: degree
    :param high: degree
    :return: image with change
    """
    total = img.shape[0] * img.shape[1]
    for i in range(img.shape[2]):
        unique, counts = np.unique(img[:, :, i], return_counts=True)
        current = 0
        for u, c in zip(unique, counts):
            if float(current) / total < low:
                low_val = u
            if float(current) / total < high:
                high_val = u
            current += c
        img[:, :, i] = np.maximum(np.minimum(img[:, :, i], high_val), low_val)
    return img
def auto_multi_scale_retinex(img, sigma_list):
    """
    Implements the retinex algorithm.
    :param img: source
    :param sigma_list: list sigma (degrees)
    :return: image with change
    """
    img = np.float64(img) + 1.0
    img_retinex = multiScaleRetinex(img, sigma_list)
```

```
    for i in range(img_retinex.shape[2]):
        unique, count = np.unique(np.int32(img_retinex[:, :, i] * 100),
return_counts=True)
        for u, c in zip(unique, count):
            if u == 0:
                zero_count = c
                break
        low_val = unique[0] / 100.0
        high_val = unique[-1] / 100.0
        for u, c in zip(unique, count):
            if u < 0 and c < zero_count * 0.1:
                low_val = u / 100.0
            if u > 0 and c < zero_count * 0.1:
                high_val = u / 100.0
                break
        img_retinex[:, :, i] = np.maximum(np.minimum(img_retinex[:, :, i],
high_val), low_val)
        img_retinex[:, :, i] = (img_retinex[:, :, i] - np.min(img_retinex[:, :,
i])) / \
                               (np.max(img_retinex[:, :, i]) -
np.min(img_retinex[:, :, i])) * 255
    img_retinex = np.uint8(img_retinex)
    return img_retinex
def multi_scale_retinex(img, sigma_list, low, high):
    """
    Implements the retinex algorithm with scales.
    :param img: source
    :param sigma_list: list sigma (degrees)
    :param low: degree
    :param high: degree
    :return: image with change
    """
    img = np.float64(img) + 1.0
    intensity = np.sum(img, axis=2) / img.shape[2]
    retinex = multiScaleRetinex(intensity, sigma_list)
    intensity = np.expand_dims(intensity, 2)
    retinex = np.expand_dims(retinex, 2)
    intensity1 = simplestColorBalance(retinex, low, high)
    intensity1 = (intensity1 - np.min(intensity1)) / (np.max(intensity1) -
np.min(intensity1)) * 255.0 + 1.0
    img_multi_scale_retinex = np.zeros_like(img)
    for y in range(img_multi_scale_retinex.shape[0]):
        for x in range(img_multi_scale_retinex.shape[1]):
            B = np.max(img[y, x])
            A = np.minimum(256.0 / B, intensity1[y, x, 0] / intensity[y, x, 0])
            img_multi_scale_retinex[y, x, 0] = A * img[y, x, 0]
            img_multi_scale_retinex[y, x, 1] = A * img[y, x, 1]
            img_multi_scale_retinex[y, x, 2] = A * img[y, x, 2]
    img_multi_scale_retinex = np.uint8(img_multi_scale_retinex - 1.0)
    return img_multi_scale_retinex
```

config.json

```
{    "sigma_list": [15, 80, 250],
    "low"  : 0.01,
    "high" : 0.99
```

Исходное и два модифицированных изображения.



2.16. Применение преобразования Хаара для подавления шумов на изображении

Информация о преобразовании Хаара

Преобразование Хаара используется для сжатия входных сигналов и изображений, в основном цветных и черно-белых с плавными переходами. Рассмотрим как происходит само преобразование.

Пусть имеется некоторое изображение. Каждой паре соседних элементов (пикселей) ставятся в соответствие два числа: сумма, делённая на корень из 2 и разность, делённая на корень из 2. Почему именно «полусумма» и «полуразность»? Дело в том, что в «реальных» изображениях яркость соседних пикселей обычно отличается на маленькую величину. Поэтому «полусумма» нам даёт усреднённое значение яркости, отфильтровывая случайные всплески. А «полуразность», в свою очередь, отфильтровывают низкие частоты. Таким

образом, преобразование Хаара делит сигнал на два: высокочастотный и низкочастотный. Что нам это даёт?

Низкочастотная составляющая несёт в себе информацию об общей форме предметов на изображении, о плавных перепадах яркости. Высокочастотная — это шум и мелкие детали. Обычно, когда мы смотрим на изображение нас интересует именно низкочастотная составляющая, а значит частью высокочастотных данных мы можем пренебречь при сжатии. То есть, таким образом мы сможем устранить шум с изображения. После этого необходимо выполнить обратное преобразование Хаара, чтобы восстановить изображение.

Алгоритм устранения шума с изображения с помощью преобразования Хаара состоит в следующем:

1) Получаем пару соседних пикселей
2) Выполняем прямое преобразование Хаара
3) Выбираем пороговые значения шума по уровням разложения
4) Производим фильтрацию
5) Реконструируем изображение(обратное преобразование Хаара)

Использованные функции

cv2.imread	Загружает изображение из указанного файла
cv2.resize	Изменение размера изображения
cv2.imshow	Используется для отображения изображения в окне
np.sqrt	Вычисляет квадратный корень
np.zeros	Возвращает массив, заполненный нулями
np.array	Создаёт массив
cv2.bitwise_or	Выполняет побитовое или
cv2.waitKey	Ожидает нажатие клавиши

Листинг программы 1

```
import cv2
import numpy as np
```

```
def haar(n, x):
    u = x.copy()
    s = np.sqrt(2.0)
    v = np.zeros(n, dtype=np.float64)
    k = 1
    while k * 2 <= n:
        k = k * 2
    while 1 < k:
        k = k // 2

        v[0:k] = (u[0:2 * k - 1:2] + u[1:2 * k:2]) / s
        v[k:2 * k] = (u[0:2 * k - 1:2] - u[1:2 * k:2]) / s

        u[0:2 * k] = v[0:2 * k].copy()

    return u

def haar_inverse(n, x):
    u = x.copy()
    s = np.sqrt(2.0)
    v = np.zeros(n)
    k = 1
    while k * 2 <= n:
        v[0:2 * k - 1:2] = (u[0:k] + u[0 + k:k + k]) / s
        v[1:2 * k:2] = (u[0:k] - u[0 + k:k + k]) / s

        u[0:2 * k] = v[0:2 * k].copy()
        k = k * 2

    return u

img = cv2.imread('test2.png')
img = cv2.resize(img, (403, 52))
height, width, channels = img.shape

const = 230
cv2.imshow(' Before ', img)

for k in range(0, 1):
    for j in range(0, width - 1):
        i = 0
        while i < height:
            pixel1 = img[i, j, 0]
            pixel2 = img[i, j + 1, 0]
            x = np.array([pixel1, pixel2])
            x = haar(2, x)

            a = x[1]
            if a < const:
                img[i, j + 1, 0] = 0
            if a > const:
                img[i, j + 1, 0] = x[1]

            x = haar_inverse(2, x)

            img[i, j, 0] = x[0]

            pixel1 = img[i, j, 1]
            pixel2 = img[i, j + 1, 1]
            x = np.array([pixel1, pixel2])
```

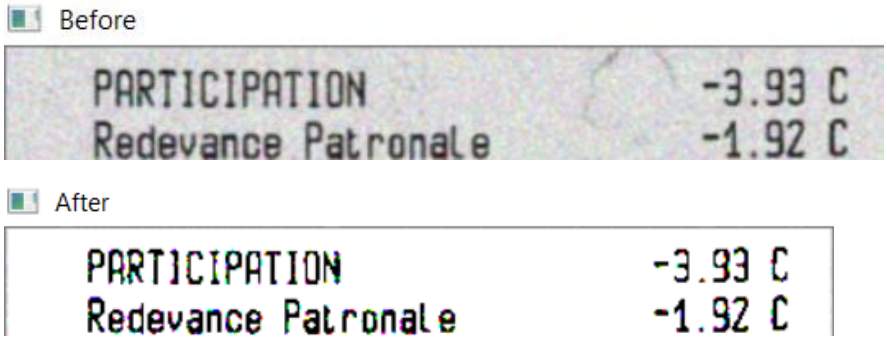
```
            x = haar(2, x)

            a = x[1]
            if a < const:
                img[i, j + 1, 1] = 0
            if a > const:
                img[i, j + 1, 1] = x[1]

            x = haar_inverse(2, x)
            img[i, j, 1] = x[0]

            pixel1 = img[i, j, 2]
            pixel2 = img[i, j + 1, 2]
            x = np.array([pixel1, pixel2])
            x = haar(2, x)

            a = x[1]
            if a < const:
                img[i, j + 1, 2] = 0
            if a > const:
                img[i, j + 1, 2] = x[1]

            x = haar_inverse(2, x)
            img[i, j, 2] = x[0]

            i = i + 1

opened_mask = cv2.imread('test2.png')
opened_mask = cv2.resize(opened_mask, (403, 52))

res = cv2.bitwise_or(opened_mask, img)
cv2.imshow(' After ', res)
cv2.waitKey()
```

Листинг программы 2

```
import cv2
import numpy as np
from skimage import color, data, img_as_float
from skimage.restoration import (denoise_wavelet, estimate_sigma)

# Считываем изображение и делаем его черно-белым
image = cv2.imread("test.png")
image = img_as_float(image)
image = color.rgb2gray(image)

# Функция преобразования Хаара
def haarTransformation(val):
    alpha = val/8
    beta = ( 1.0 - alpha )
    denoised_image = denoise_wavelet(image, sigma = 0.08, wavelet = "haar",
rescale_sigma=True) #Создаем преобразованную матрицу
    result = cv2.addWeighted(image, alpha, denoised_image, beta, 0.0) # Получаем
итоговое изображение
    cv2.imshow("Image", result) # Выводим изображение на экран

cv2.namedWindow('Image')
cv2.createTrackbar("Noise level", "Image", 0, 30, haarTransformation) # Создаем
ползунок
haarTransformation(0) # Вызываем преобразование Хаара
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Результат



2.17. Пороговая обработка

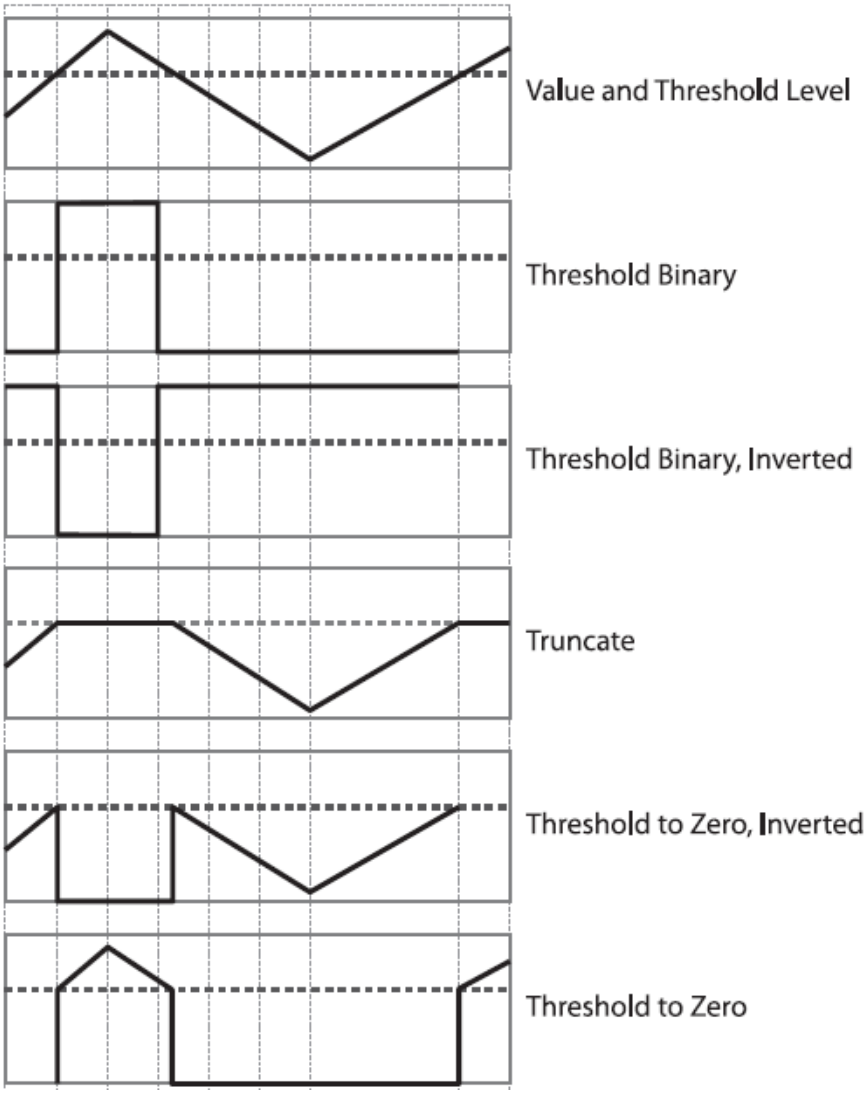
Пороговая обработка очень проста, если значение пикселя превышает пороговое значение, то он окрашивается в белый цвет, в обратном случае он окрашивается в черный цвет.

Существуют разные пороговые методы:

- **cv2.THRESH_BINARY;**
- **cv2.THRESH_BINARY_INV;**
- **cv2.THRESH_TRUNC;**
- **cv2.THRESH_TOZERO;**
- **cv2.THRESH_TOZERO_INV;**

На фотографии представлены их различия.

Программа показывает работу пороговой обработки с глобальным порогом, с несколькими порогами, а также с переменным порогом. Использован метод обработки cv2.THRESH_BINARY. Показана разница между разными порогами обработки.



В первой программе представлена пороговая обработка с глобальным порогом, адаптивным порогом и адаптивным гауссовским порогом, все используют обычный бинарный порог.

•Глобальный порог с порогом для классификации значений пикселей равным 100

•Адаптивный порог. Считается среднее значение из 9 пикселей вокруг, благодаря этому можно получить лучшие результаты для изображений с разной яркостью, так как значение будет меняться для разных областей изображения, но для лучшего качества нужно подбирать размер области для высчитывания среднего значения, и число, которое

вычитается из среднего порога, поэтому придется делать несколько проверок.

•Адаптивный метод Гаусса. Вместо среднего значения используется средневзвешенная сумма всех пикселей в области размером 9 пикселей. Вес определяется методом Гаусса. Минусы те же что и у обычного адаптивного метода.

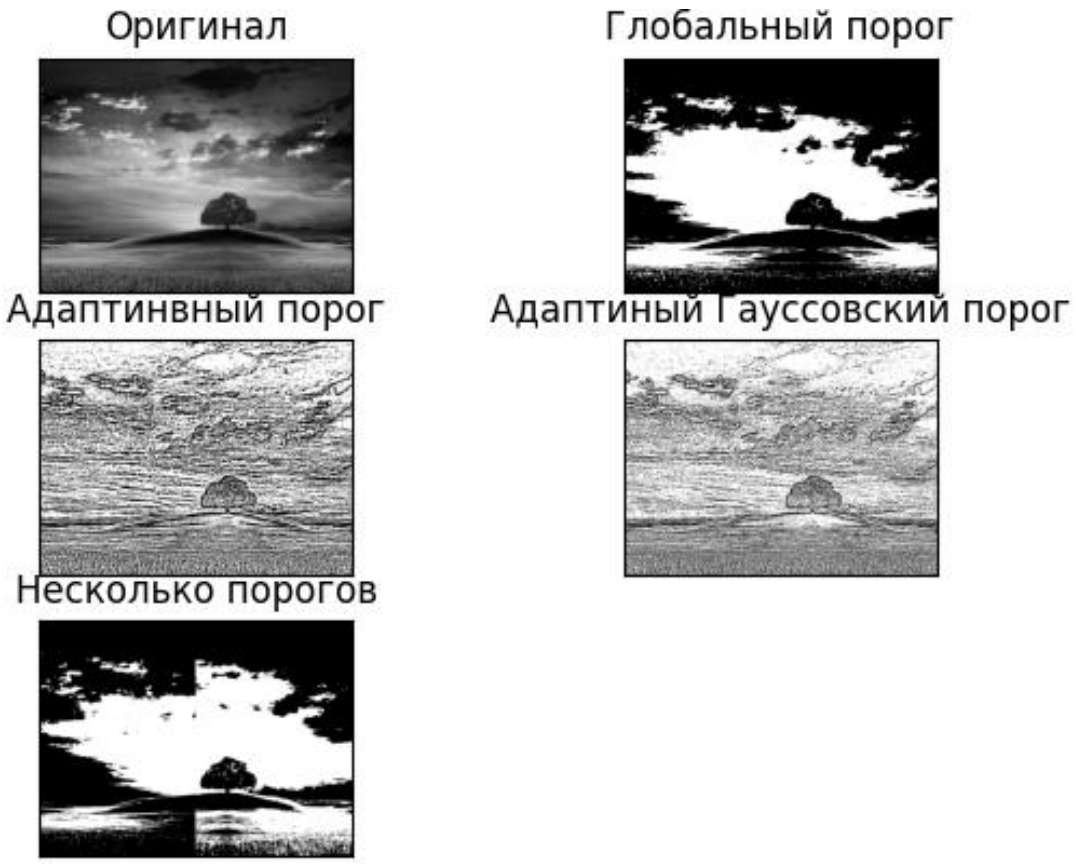
•Обработка изображения несколькими порогами. Изображение было разделено на 2 части, каждая из частей прошла пороговую обработку, а потом изображения были склеены обратно методом np.concatenate().

Листинг программы1

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot

img = cv.imread('pict.png', 0)
height, width = img.shape[:2]

# обрезание левой половины изображения
start_row, start_col = int(0), int(0)
end_row, end_col = int(height), int(width * .5)
cropped_top = img[start_row:end_row, start_col:end_col]
ret, th4 = cv.threshold(cropped_top, 120, 255, cv.THRESH_BINARY)

# обрезание правой половины изображения и её преобразование
start_row, start_col = int(0), int(width * .5)
end_row, end_col = int(height), int(width)
cropped_bot = img[start_row:end_row, start_col:end_col]
ret, th5 = cv.threshold(cropped_bot, 70, 255, cv.THRESH_BINARY)

# соединение обратно по горизонтали
th6 = np.concatenate((th4, th5), axis=1)

# глобальные порог
ret, th1 = cv.threshold(img, 100, 255, cv.THRESH_BINARY)

# адаптивный порог
th2 = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY,
7, 2)
# гауссовский порог
th3 = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C,
cv.THRESH_BINARY, 7, 2)
titles = ['Оригинал', 'Глобальный порог',          'Адаптивный порог', 'Адаптивный
Гауссовский порог', 'Несколько порогов']
images = [img, th1, th2, th3, th6]
for i in range(5):
    pyplot.subplot(3, 2, i + 1), pyplot.imshow(images[i], 'gray')
    pyplot.title(titles[i])
```

```
    pyplot.xticks([]), pyplot.yticks([])
pyplot.show()
```

Результат:



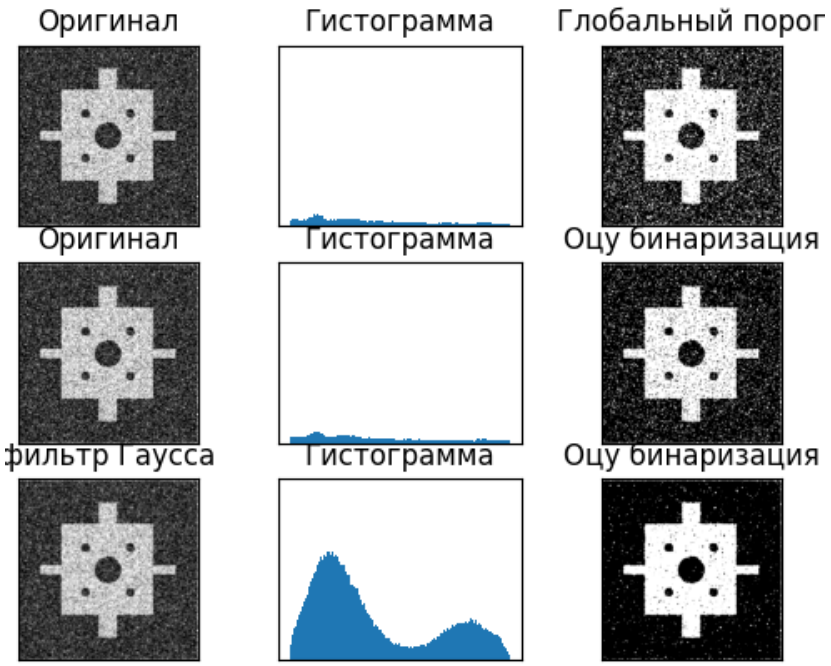
Во второй программе представлена пороговая обработка с использованием дополнительного флага cv.THRESH_OTSU, благодаря которому очень удобно использовать пороговую обработку для бимодальных изображений (гистограммы таких изображений имеют два пика).

В этом примере видно насколько лучше работает Оцу бинаризация для бимодальных изображений, фильтр Гаусса увеличил эффект зернистости на рисунке, что привело к тому, что гистограмма стала иметь два пика, что привело к улучшению качества рисунка по сравнению с обычным глобальным порогом и бинаризацией Оцу не с бимодальным изображением.

Листинг программы 2

```
import cv2 as cv
from matplotlib import pyplot

img = cv.imread('picture.png', 0)
# Фиксированный порог
ret1, th1 = cv.threshold(img, 100, 255, cv.THRESH_BINARY)
# Отсу порог
ret2, th2 = cv.threshold(img, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
# Отсу порог после гауссовой фильтрации
blur = cv.GaussianBlur(img, (5, 5), 0)
ret3, th3 = cv.threshold(blur, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
# Нарисуйте все изображения и их гистограммы
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Оригинал', 'Гистограмма изображения', 'Глобальный порог',
          'Оригинал', 'Гистограмма изображения', "Оцу бинаризация",
          'Оригинал', 'Гистограмма изображения', "Оцу бинаризация"]
for i in range(3):
    pyplot.subplot(3, 3, i * 3 + 1), pyplot.imshow(images[i * 3], 'gray')
    pyplot.title(titles[i * 3]), pyplot.xticks([]), pyplot.yticks([])
    pyplot.subplot(3, 3, i * 3 + 2), pyplot.hist(images[i * 3].ravel(), 256)
    pyplot.title(titles[i * 3 + 1]), pyplot.xticks([]), pyplot.yticks([])
    pyplot.subplot(3, 3, i * 3 + 3), pyplot.imshow(images[i * 3 + 2], 'gray')
    pyplot.title(titles[i * 3 + 2]), pyplot.xticks([]), pyplot.yticks([])
pyplot.show()
```

Результат:



2.18. Задача сопоставления изображений. Детекторы углов Харриса, LOG, DOG

Для решения задачи сопоставления изображений необходимо нахождение графических признаков, по которым можно распознавать образы и проводить дальнейшие сравнения. Одним из основных признаков являются особые точки. Под особенной точкой понимают точку изображения, локальные окрестности которых обладают некоторыми отличительными особенностями в сравнении с окрестностями остальных точек изображения.

Основные свойства особых точек:

- *Отличимость (distinctness)* – особая точка должна явно выделяться на фоне и быть отличимой (уникальной) в своей окрестности.
- *Инвариантность (invariance)* – определение особой точки должно быть независимо к аффинным преобразованиям.
- *Стабильность (stability)* – определение особой точки должно быть устойчиво к шумам и ошибкам.
- *Уникальность (uniqueness)* – кроме локальной отличимости, особая точка должна обладать глобальной уникальностью для улучшения различимости повторяющихся паттернов.
- *Интерпретируемость (interpretability)* – особые точки должны определяться так, чтобы их можно было использовать для анализа соответствий и выявления интерпретируемой информации из изображения.

Детектор углов Харриса позволяет находить особые точки, расположенные на углах – точках, которые формируются из двух или более граней. В области вокруг угла у градиента изображения преобладают два доминирующих направления, что делает их

различимыми. Данный детектор наиболее эффективен для нахождения L-углов:

Еще одним важным графическим признаком является нахождение контуров объектов на изображении. Рассмотрим детекторы DoG (Difference of Gaussians) и LoG (Laplacian of Gaussians).

Лапласовский фильтр (LoG) — это один из детекторов границ. Лапласиан представляет собой сумму частных производных второго порядка, при переходе через границу, он меняет знак на противоположный. Другими словами, фильтр Лапласа выделяет области, в которых интенсивность пикселей резко меняется.

$$\nabla^2 f = \Delta f = \left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\right).$$

Разность Гауссианов также применяется в качестве детектора границ. Принцип работы этого детектора основан на нахождении разности между двумя изображениями, размытыми с разным размытием по Гауссу, которое удаляет высокочастотные фрагменты изображения. В итоге получается, что из-за разницы в размытии, при их вычитании низкочастотные фрагменты сокращаются и остаются только высокочастотные, которые соответствуют границам объекта.

Вдобавок к этим методам существует попиксельное сравнение изображений для установления соответствия или нахождения различий.

Список используемых функций

cv2.imread; Image.open	Загружает изображение из указанного файла

cv2.GaussianBlur	Осуществляет размытие по Гауссу
cv2.imshow	Используется для отображения изображения в окне
cv2.Laplasian	Аналог оператора Лапласа для двумерного случая
cv2.cvtColor	Изменяет световую схему изображения
cv2.cornerHarris	Находит угловые особенные точки
Image.thumbnail	Изменяет размер изображения
ImageChops.difference	Попиксельно сравнивает изображения

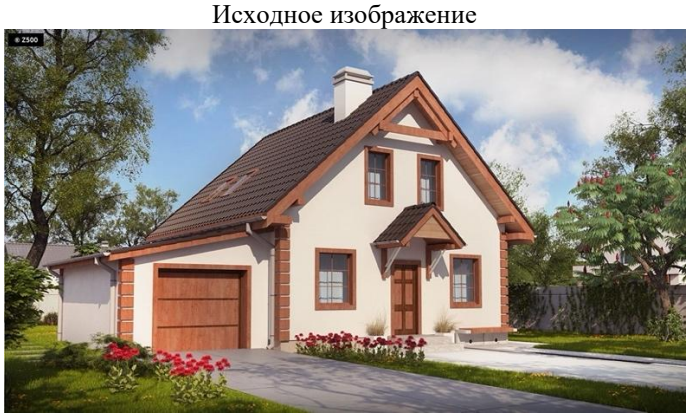
Листинг программы:

```
import cv2
import numpy as np
from PIL import Image, ImageChops

#  Детекторы
img = cv2.imread('Path/house.jpg')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Применение 3x3 и 5x5 размытия по Гауссу
low_gauss = cv2.GaussianBlur(gray_img, (15, 15), 0)
high_gauss = cv2.GaussianBlur(gray_img, (17, 17), 0)

# Вычисление DoG
dog = low_gauss - high_gauss
cv2.imshow("DoG Result", dog)

# Вычисление LoG
laplacian = cv2.Laplacian(low_gauss, cv2.CV_64F)
laplacian1 = laplacian / laplacian.max()
cv2.imshow("LoG Result", laplacian1)

# Детектор углов Харриса
gray_img = np.float32(gray_img)
dest = cv2.cornerHarris(gray_img, 2, 5, 0.07)

# Увеличение размера маркеров
kernel = np.ones((4, 4), 'uint8')
dest = cv2.dilate(dest, kernel, None)

# Отрисовка маркеров
img[dest > 0.008 * dest.max()] = [0, 0, 255]

cv2.imshow("Corners detector Result", img)

cv2.waitKey(0)
cv2.destroyAllWindows()

# Попиксельное сравнение изображений
image_1 = Image.open('Path/inter.jpg')
image_2 = Image.open('Path/inter2.jpg')

# Приведение изображений к одному размеру
size = [400, 300]
image_1.thumbnail(size)
```

```
image_2.thumbnail(size)

result = ImageChops.difference(image_1, image_2)

if result == None:
    print("images matched!")
else:
    result.show()
    result.save('result.jpg')

cv2.destroyAllWindows()
```

Примеры применения программы:

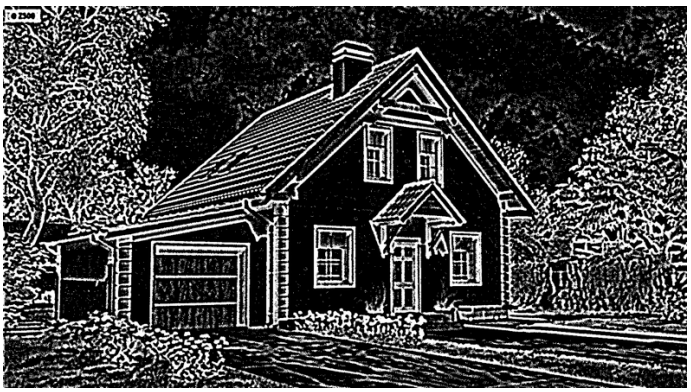
Исходное изображение



Детектор углов Харриса

Детектор контуров - LoG

Детектор контуров – DoG



2.19. Сравнение изображений и генерация картинки отличий

Программа ищет различия в двух изображениях. Реализованы две функции, выполняющие поиск различий разными подходами и дающие ответ в разном виде. Входные данные подаются в виде двух файлов (которые преобразуются до размеров 640x480, чтобы избежать проблем со слишком большими или слишком маленькими изображениями). Результат работы программы отображается на экран и записывается на диск. Виды отображения:

1. Для первой функции (*changesOnCanvas*): изображение, на котором на черном фоне показаны различия между исходными изображениями (положение различий на изображении сохранено)
2. Для второй функции (*changesWithAreas*): копию второго изображения, на котором красными прямоугольниками выделены различия между ним и первым изображением

Используемые функции:

numpy	
numpy.zeros_like(img2, numpy.uint8)	Создаем массив с той же структурой, что и img2, но заполненный восьмибитными целочисленными беззнаковыми нулями

imutils	
imutils.resize(img, height=640, width=480)	Изменяем размер исходного изображения на 640x480 пикселей, чтобы избежать отображения слишком больших или слишком маленьких изображений на экране
imutils.grab_contours(cnts)	Преобразуем возвращаемое значение функции

	cv2.findContours, удаляя из него иерархию и возвращая только сами контуры

cv2	
cv2.imread(imagePath)	Читаем изображение с диска
cv2.absdiff(img1, img2)	Находим абсолютную разность двух изображений, чтобы обнаружить различия между ними
cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)	Преобразуем полученное в результате работы функции cv2.absdiff дифференциальное изображение в градации серого
cv2.imshow(winname, img)	Отображаем изображение img на экране в окне с названием winname
cv2.imwrite(filename, img)	Записываем изображение img на диск под именем filename
cv2.waitKey(0)	Ожидаем нажатия любой клавиши
cv2.destroyWindow(winname)	Уничтожаем окно с названием winname
cv2.dilate(gray.copy(), None, iterations=i + 1)	Применяем морфологический фильтр к изображению в градациях серого, чтобы увеличить различия на нем
cv2.threshold(dilated, 3, 255, cv2.THRESH_BINARY)	Применяем порог фиксированного уровня к многоканальному массиву для получения двухуровневого (двоичного) изображения из результата работы функции cv2.cvtColor (изображения в градациях серого)
cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)	Ищем контуры различий на двухканальном изображении, полученном из функции cv2.threshold, получая склеенные контуры без группировки
cv2.boundingRect(c)	Получаем значения для построения прямоугольников по контурам различий в изображениях
cv2.rectangle(new, (x, y), (x + w, y + h), (0, 0, 255), 2)	Добавляем на исходное изображение красные прямоугольники по контуру вокруг обнаруженных различий

Собственные функции	
def compareOnCanvas(imagePath1, imagePath2)	Читаем изображения с диска, преобразуем к размеру 640x480 пикселей, обнаруживаем различия, отображаем их в новом окне на черном фоне и записываем на диск в файл changesOnCanvas.png
def compareWithAreas(imagePath1, imagePath2)	Читаем изображения с диска, преобразуем к размеру 640x480 пикселей, обнаруживаем различия, отображаем исходное изображение в новом окне, выделяя различия красными прямоугольниками, и записываем на диск в файл changesWithAreas.png

Листинг программы

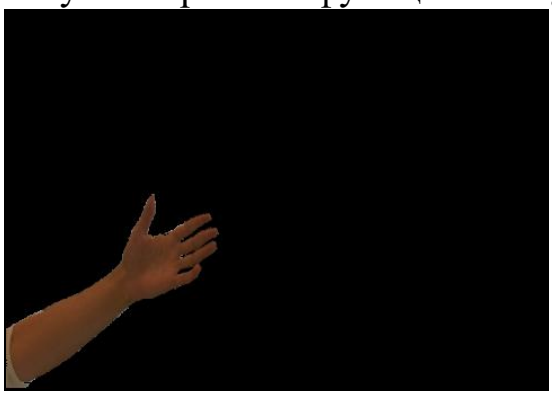
```
import cv2
import imutils
```

```
import numpy


def compareOnCanvas(imagePath1, imagePath2):
    img1 = cv2.imread(imagePath1)
    img2 = cv2.imread(imagePath2)

    img1 = imutils.resize(img1, height=640, width=480)
    img2 = imutils.resize(img2, height=640, width=480)

    diff = cv2.absdiff(img1, img2)
    mask = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)

    th = 1
    imask = mask > th

    canvas = numpy.zeros_like(img2, numpy.uint8)
    canvas[imask] = img2[imask]

    cv2.imshow("Differences on canvas", canvas)
    cv2.imwrite("changesOnCanvas.png", canvas)
    cv2.waitKey(0)
    cv2.destroyWindow("Differences on canvas")


def compareWithAreas(imagePath1, imagePath2):
    original = cv2.imread(imagePath1)
    new = cv2.imread(imagePath2)

    original = imutils.resize(original, height=640, width=480)
    new = imutils.resize(new, height=640, width=480)

    diff = original.copy()
    cv2.absdiff(original, new, diff)

    gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)

    for i in range(0, 3):
        dilated = cv2.dilate(gray.copy(), None, iterations=i + 1)

    (T, thresh) = cv2.threshold(dilated, 3, 255, cv2.THRESH_BINARY)

    cnts = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)

    for c in cnts:
        (x, y, w, h) = cv2.boundingRect(c)
        cv2.rectangle(new, (x, y), (x + w, y + h), (0, 0, 255), 2)

    cv2.imshow("Differences with areas", new)
    cv2.imwrite("changesWithAreas.png", new)
    cv2.waitKey(0)
    cv2.destroyWindow("Differences with areas")


im1Path = "original.png"
im2Path = "withArm.png"

compareOnCanvas(im1Path, im2Path)
compareWithAreas(im1Path, im2Path)
```

Пример работы программы

Исходные изображения

Результат работы функций changesOnCanvas и changesWithAreas



2.20. Фильтры в OpenCV

Программа имеет графический интерфейс, который позволяет: запустить поток видео с веб-камеры, и применять различные фильтры к этому потоку в реальном времени. В процессе работы в цикле получаем кадр с вебкамеры, считываем значение позлунков и в зависимости от них, включаем/выключаем нужный фильтр.

Используемые функции

def apply_color_overlay (frame, intensity=0.2, blue=0, green=0, red=0)	Создаём наложение, используя выходные RGB overlay = np.full((frame_h, frame_w, 4), color_bgra, dtype='uint8'). Потом применяем наложение к кадру cv2.addWeighted(overlay, intensity, frame, 1.0, 0, frame)

def verify_alpha_channel (frame)	Проверка наличия alpha канала, иначе преобразовываем в BGRA
def apply_invert (frame)	Инвертирует каждый бит кадра, используя cv2.bitwise_not(frame)
def alpha_blend (frame_1, frame_2, mask)	Передаём основной фрейм, маску наложения (как фрейм) и саму маску (число). Создаём альфа на основе mask. Дальше смешиваем всё в один frame cv2.convertScaleAbs(frame_1 * (1 - alpha) + frame_2 * alpha)
def apply_circle_focus_blur (frame, intensity=0.2)	Выбираем сначала координаты и радиус блюра, который применяется с помощью Гауссовской маски cv2.GaussianBlur (mask, (21, 21), 11)
def apply_portrait_mode (frame)	Удаляем шум с помощью cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY) и замаливаем чуть-чуть всю картинку cv2.GaussianBlur (frame, (21, 21), 11)
cvtColor (blur2, cv2.COLOR_BGR2BGRA)	преобразует изображение из одного цветового пространства в другое
threshold (gray, 120, 255, cv2.THRESH_BINARY)	функция применяет порог фиксированного уровня к многоканальному массиву. Обычно используется для получения двухуровневого (двоичного) изображения из изображения в градациях серого
GaussianBlur (frame, (21, 21), 11)	Размытие по Гауссу. В этом методе вместо прямоугольного фильтра используется гауссово ядро. Мы должны указать ширину и высоту ядра, которые должны быть положительными и нечетными
convertScaleAbs (src)	для каждого элемента массива (пикселя в кадре) выполняет последовательно 3 операции: масштабирование, взятие абсолютного значение, преобразование в беззнаковый 8битный тип.
addWeighted (overlay, intensity, frame, 1.0, 0, frame)	накладывает 2 кадра друг на друга, используя веса для каждого из них

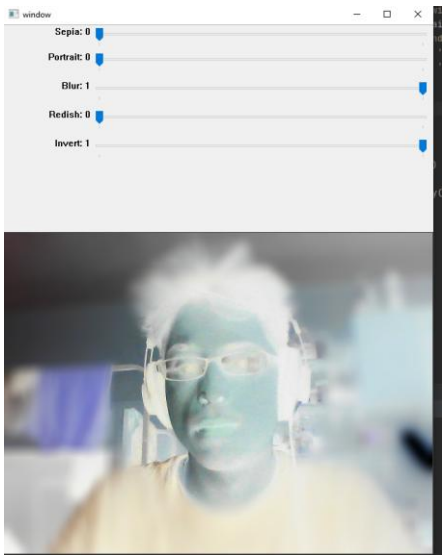
Листинг программы

```
import cv2
import numpy as np

if __name__ == '__main__':
    def nothing(*arg):
        pass

cap = cv2.VideoCapture(0)

cv2.namedWindow("window")

switchSepia = 'Sepia'
```

```
cv2.createTrackbar(switchSepia, 'window', 0, 1, nothing)
switchPortrait = 'Portrait'
cv2.createTrackbar(switchPortrait, 'window', 0, 1, nothing)
switchBlur = 'Blur'
cv2.createTrackbar(switchBlur, 'window', 0, 1, nothing)
switchRedish = 'Redish'
cv2.createTrackbar(switchRedish, 'window', 0, 1, nothing)
switchInvert = 'Invert'
cv2.createTrackbar(switchInvert, 'window', 0, 1, nothing)

def apply_invert(frame):
    return cv2.bitwise_not(frame)


def verify_alpha_channel(frame):
    try:
        frame.shape[3]  # 4th position
    except IndexError:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2BGRA)
    return frame

def apply_color_overlay(frame,
                        intensity=0.2,
                        blue=0,
                        green=0,
                        red=0):
    frame = verify_alpha_channel(frame)
    frame_h, frame_w, frame_c = frame.shape
    color_bgra = (blue, green, red, 1)
    overlay = np.full((frame_h, frame_w, 4), color_bgra, dtype='uint8')
    cv2.addWeighted(overlay, intensity, frame, 1.0, 0, frame)
    frame = cv2.cvtColor(frame, cv2.COLOR_BGRA2BGR)
    return frame

def apply_sepia(frame, intensity=0.5):
    blue = 20
    green = 66
    red = 112
    frame = apply_color_overlay(frame,
                            intensity=intensity,
                            blue=blue, green=green, red=red)
    return frame

def alpha_blend(frame_1, frame_2, mask):
    alpha = mask / 255.0
    blended = cv2.convertScaleAbs(frame_1 * (1 - alpha) + frame_2 * alpha)
    return blended


def apply_circle_focus_blur(frame, intensity=0.2):
    frame = verify_alpha_channel(frame)
    frame_h, frame_w, frame_c = frame.shape
    y = int(frame_h / 2)
    x = int(frame_w / 2)
    radius = int(x / 2)  # int(x/2)
    center = (x, y)
    mask = np.zeros((frame_h, frame_w, 4), dtype='uint8')
    cv2.circle(mask, center, radius, (255, 255, 255), -1, cv2.LINE_AA)
    mask = cv2.GaussianBlur(mask, (21, 21), 11)
    blured = cv2.GaussianBlur(frame, (21, 21), 11)
    blended = alpha_blend(frame, blured, 255 - mask)
```

```
    frame = cv2.cvtColor(blended, cv2.COLOR_BGRA2BGR)
    return frame


def apply_portrait_mode(frame):
    frame = verify_alpha_channel(frame)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    _, mask = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
    mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGRA)
    blured = cv2.GaussianBlur(frame, (21, 21), 11)
    blended = alpha_blend(frame, blured, mask)
    frame = cv2.cvtColor(blended, cv2.COLOR_BGRA2BGR)
    return frame


while True:
    ret, frame = cap.read()

    sepia = cv2.getTrackbarPos(switchSepia, 'window')
    portrait = cv2.getTrackbarPos(switchPortrait, 'window')
    blur = cv2.getTrackbarPos(switchBlur, 'window')
    redish = cv2.getTrackbarPos(switchRedish, 'window')
    invert = cv2.getTrackbarPos(switchInvert, 'window')

    if sepia == 1:
        frame = apply_sepia(frame.copy())
    if portrait == 1:
        frame = apply_portrait_mode(frame)
    if blur == 1:
        frame = apply_circle_focus_blur(frame)
    if redish == 1:
        frame = apply_color_overlay(frame.copy(), intensity=.5, red=230, blue=10)
    if invert == 1:
        frame = apply_invert(frame)

    cv2.imshow('window', frame)

    if cv2.waitKey(20) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Пример работы


window
Sepia: 0
Portrait: 0
Blur: 1
Redish: 0
Invert: 1

Глава 3. Сегментация изображений

3.1. Обзор методов сегментации изображений

В компьютерном зрении, сегментация — это процесс разделения цифрового изображения на несколько сегментов (множество пикселей, также называемых суперпикселями). Цель сегментации заключается в упрощении и/или изменении представления изображения, чтобы его было проще и легче анализировать. Результатом сегментации изображения является множество сегментов, которые вместе покрывают всё изображение, или множество контуров, выделенных из изображения. Все пиксели в сегменте похожи по некоторой характеристике или вычисленному свойству, например, по цвету, яркости или текстуре (в данной работе предлагается сделать сегментацию, основанную на цвете). Более точно, сегментация изображений — это процесс присвоения таких меток каждому пикселю изображения, что пиксели с одинаковыми метками имеют общие визуальные характеристики. Соседние сегменты значительно отличаются по этой характеристике.

Для сегментации изображений было разработано несколько универсальных алгоритмов и методов. Так как общего решения для задачи сегментации изображений не существует, часто эти методы приходится совмещать со знаниями из предметной области, чтобы эффективно решать эту задачу в её предметной области. Различные алгоритмы для проведения сегментации:

- Выделение краёв
- Методы разрастания областей
- Методы разреза графа
- Сегментация методом водораздела

• Сегментация с помощью модели

• Многомасштабная сегментация и другие

Выделение краев - хорошо изученная область в обработке изображений. Границы и края областей сильно связаны, так как часто существует сильный перепад яркости на границах областей. Поэтому методы выделения краёв используются как основа для другого метода сегментации. Обнаруженные края часто бывают разорванными. Но чтобы выделить объект на изображении, нужны замкнутые границы области.

Методы разреза графа могут быть эффективно применены для сегментации изображений. В этих методах изображение представляется как взвешенный неориентированный граф. Обычно пиксель или группа пикселей ассоциируется вершиной, а веса рёбер определяют (не)похожесть соседних пикселей. Затем граф (изображение) разрезается согласно критерию, созданному для получения «хороших» кластеров. Каждая часть вершин (пикселей), получаемая этими алгоритмами, считается объектом на изображении. Некоторые популярные алгоритмы этой категории — это нормализованные разрезы графов, случайное блуждание, минимальный разрез, изопериметрическое разделение

В сегментации методом водораздела рассматривается абсолютная величина градиента изображения как топографической поверхности. Пиксели, имеющие наибольшую абсолютную величину градиента яркости, соответствуют линиям водораздела, которые представляют границы областей. Вода, помещённая на любой пиксель внутри общей линии водораздела, течёт вниз к общему локальному минимуму яркости. Пиксели, от которых вода стекается к общему минимуму, образуют водосбор, который представляет сегмент.

Сегментация с помощью модели

Основное предположение этого подхода — то, что интересующие структуры или органы имеют повторяющиеся геометрические формы. Следовательно, можно найти вероятностную модель для объяснения изменений формы органа и затем, сегментируя изображение, накладывать ограничения, используя эту модель как априорную. Такое задание включает в себя (i) приведение тренировочных примеров к общей позе, (ii) вероятностное представление изменений приведённых образцов и (iii) статистический вывод для модели и изображения. Современные методы в литературе для сегментации, основанной на знании, содержат активные модели формы и внешности, активные контуры, деформируемые шаблоны и методы установления уровня.

3.2. Сегментация на основе цвета

Сегментацию можно разделить на «полную» (сегментация в привычном понимании) и «неполную» - выделение какого-то определенного цвета на изображении.

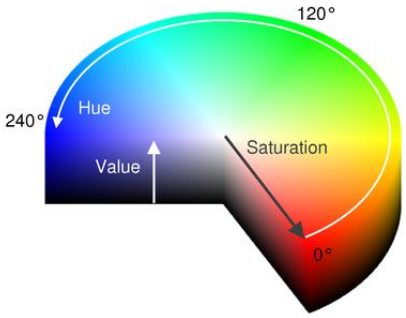
В OpenCV цветное изображение хранится палитре BGR. Определять цвет в BGR не очень удобно, поэтому переводят изображение в формат HSI.

HSI означает Hue, Saturation, Intensity, где:

Hue — цветовой тон, т.е. оттенок цвета;

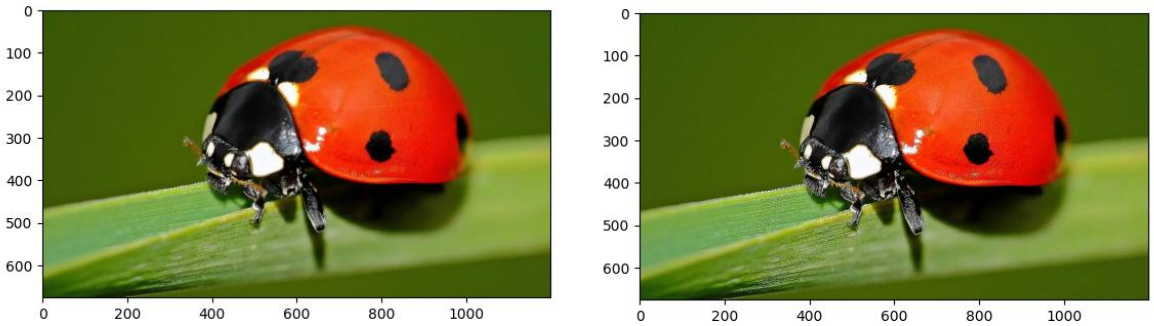
Saturation — насыщенность. Чем выше этот параметр, тем «чище» будет цвет, а чем ниже, тем ближе он будет к серому;

Intensity (value) — значение (яркость) цвета. Чем выше значение, тем ярче будет цвет (но не белее). А чем ниже, тем темнее (0% — черный)

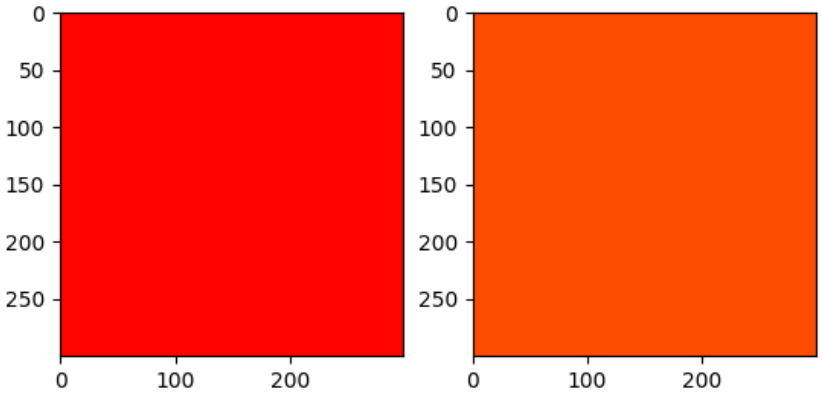


Ниже описан алгоритм сегментации по цвету с примерами изображений.

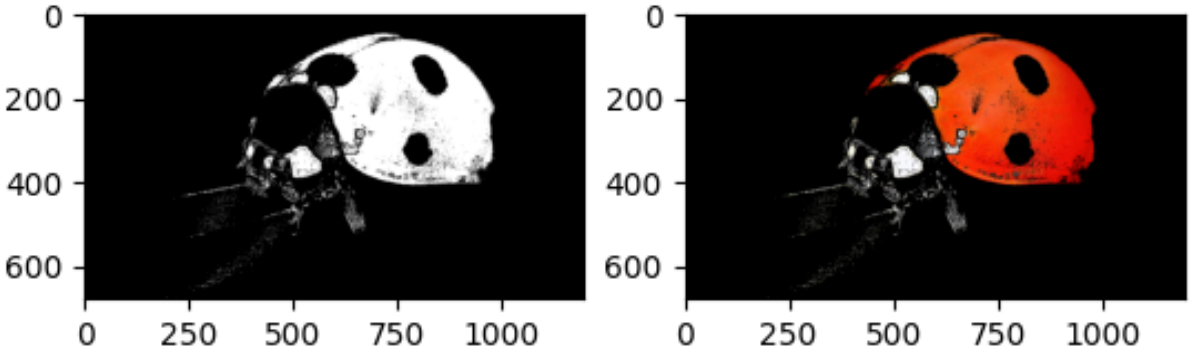
1. Ввод изображения, перевод из цветового пространства BGR в RGB
2. С помощью матрицы ядра для повышения резкости была увеличена четкость изображения



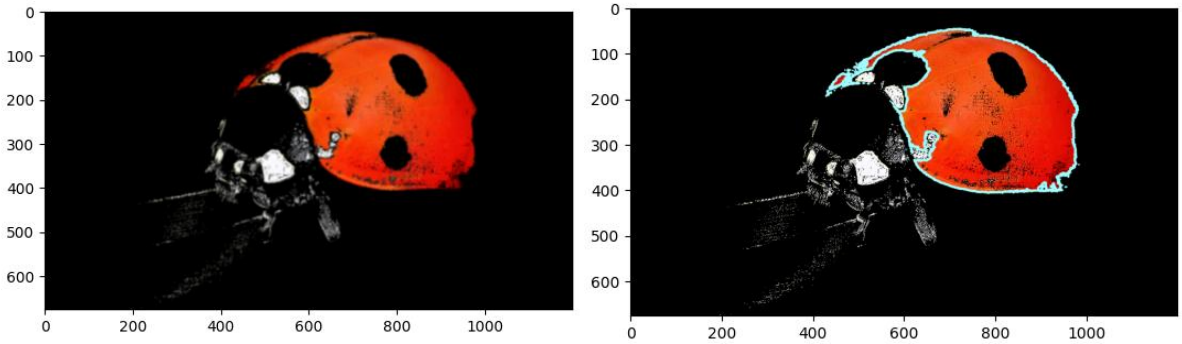
3. Далее перевели изображение в цветовое пространство HSV
4. Выделили 2 оттенка в HSV, между которыми будет наш цветовой регион, и вывели их на экран



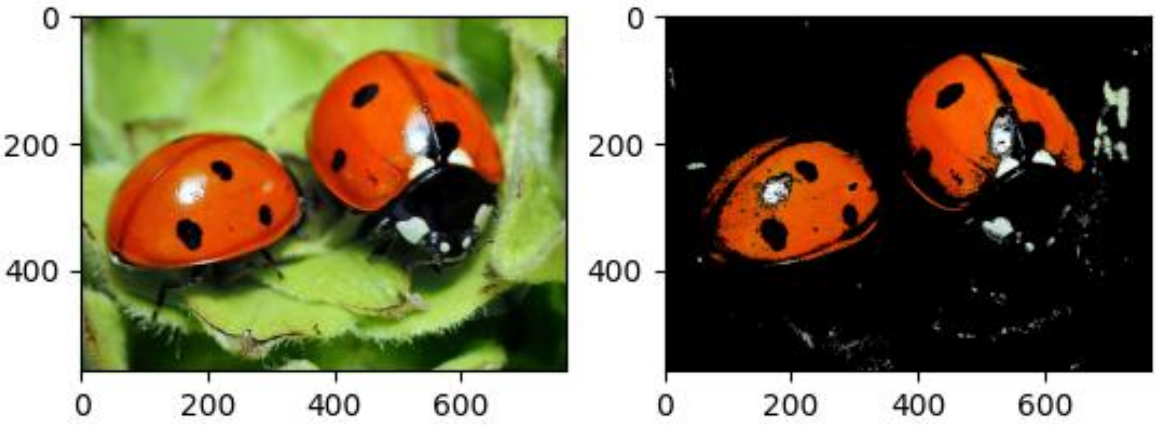
5. Кроме того, были выделены и границы региона для белых оттенков
6. С помощью функции inRange получили маску для цветного региона *mask* и для белого *mask_white*. Сложив полученные маски, наложили финальную на изображение.

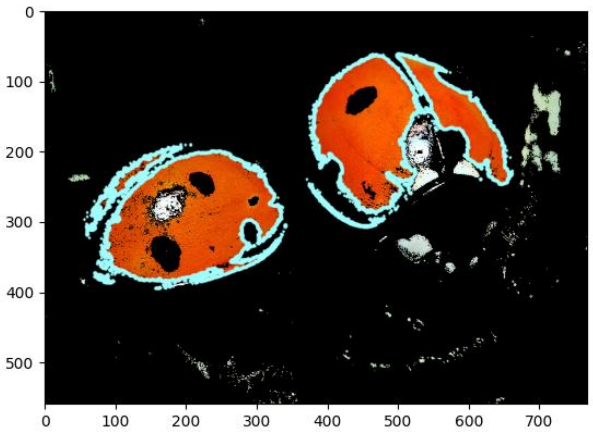


7. Чтобы уменьшить детализацию применили размытие по Гауссу
8. На основе полученного сегментированного изображения выделим контур красного региона (тела божьей коровки)



9. Проверили на другом изображении, насколько хорошо обобщает выбранная сегментация других божьих коровок. Также выделили края цветных регионов





Использованные функции

cv2.imread	Считывание изображения
cv2.cvtColor	Конвертация изображения из одного цветового пространства в другое
cv2.filter2D	Фильтрация изображения с помощью свертки между исходным изображением и ядром (определяет действие фильтрация – в данном случае резкость)
cv2.inRange	Принимает на вход изображение и диапазон цветового региона. Возвращает двоичную маску размером изображения, где значения 1 указывают значения в диапазоне, а нулевые значения указывают значения снаружи (черно-белое изображение, где белым выделены пиксели, цвета которых попадали в диапазон, а черным – с цветом вне требуемого).
cv2.bitwise_and	Вычисляет для каждого элемента побитовое соединение двух массивов – для наложения маски на изображение
cv2.GaussianBlur	Размытие изображение по Гауссу
cv2.findContours	Функция для поиска контуров findContours(кадр, режим_группировки, метод_упаковки) В качестве режима: CV_RETR_TREE — группирует контуры в многоуровневую иерархию В качестве метода: CV_CHAIN_APPROX_SIMPLE — склеивает все горизонтальные, вертикальные и диагональные контуры
cv2.drawContours	Функция для отображения контуров

Листинг программы

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

ladybug = cv2.imread('image.jpg')
ladybug = cv2.cvtColor(ladybug, cv2.COLOR_BGR2RGB)
plt.imshow(ladybug)
plt.show()

# Повышение резкости
kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
ladybug = cv2.filter2D(ladybug, -1, kernel)
```

```
plt.imshow(ladybug)
plt.show()

hsv_ladybug = cv2.cvtColor(ladybug, cv2.COLOR_RGB2HSV)

# Цветовые границы региона
low_color_hsv = (1, 190, 200)
low_color_rgb = (255, 4, 0)

high_color_hsv = (18, 255, 255)
high_color_rgb = (255, 77, 0)

# Демонстрация двух выбранных границ
low_img = np.zeros((300, 300, 3), np.uint8)
low_img[:] = low_color_rgb
plt.subplot(1, 2, 1)
plt.imshow(low_img)

high_img = np.zeros((300, 300, 3), np.uint8)
high_img[:] = high_color_rgb
plt.subplot(1, 2, 2)
plt.imshow(high_img)
plt.show()

# Границы для белых оттенков
low_white = (0, 0, 200)
high_white = (145, 60, 255)

# Получение двоичной финальной маски
mask_white = cv2.inRange(hsv_ladybug, low_white, high_white)
mask = cv2.inRange(hsv_ladybug, low_color_hsv, high_color_hsv)
final_mask = mask + mask_white

# Применение маски
result = cv2.imread('result.jpg')
result = cv2.bitwise_and(ladybug, ladybug, result, final_mask)

# Маска и исходное изображение с маской сверху
plt.subplot(1, 2, 1)
plt.imshow(final_mask, "gray")
plt.subplot(1, 2, 2)
plt.imshow(result)
plt.show()

# Сглаживание
blur = cv2.GaussianBlur(result, (9, 9), 0)
plt.imshow(blur)
plt.show()

# Поиск краев, основанный на выбранном цветовом регионе
contours, hierarchy = cv2.findContours(mask.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
result = cv2.drawContours(result, contours, -1, (161, 255, 255), 3, cv2.LINE_AA,
hierarchy, 1)
plt.imshow(result)
plt.show()

# Проверка сегментации на другом изображении
bugs = cv2.cvtColor(cv2.imread("image2.jpg"), cv2.COLOR_BGR2RGB)
bugs = cv2.filter2D(bugs, -1, kernel)
```

```
hsv_bugs = cv2.cvtColor(bugs, cv2.COLOR_RGB2HSV)
mask_white2 = cv2.inRange(hsv_bugs, low_white, high_white)
mask2 = cv2.inRange(hsv_bugs, low_color_hsv, high_color_hsv)
final_mask2 = mask_white2 + mask2

result2 = cv2.imread('result1.jpg')
result2 = cv2.bitwise_and(bugs, bugs, result2, final_mask2)

plt.subplot(1, 2, 1)
plt.imshow(bugs)
plt.subplot(1, 2, 2)
plt.imshow(result2)
plt.show()

blur2 = cv2.GaussianBlur(result2, (9, 9), 0)
contours2, hierarchy2 = cv2.findContours(mask2.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
result2 = cv2.drawContours(result2, contours2, -1, (161, 255, 255), 3,
cv2.LINE_AA, hierarchy2, 1)
plt.imshow(result2)
plt.show()
```

3.3. Метод морфологического водораздела

Понятие водораздела основано на представлении изображения как трехмерной поверхности, где в качестве высоты используется уровень яркости пикселя. В этом случае на поверхности можно обнаружить три типа точек:

- Точки локального минимума;
- Точки, находящиеся на склоне, с которых вода сливается к центру водоема;
- Точки, находящиеся на гребне возвышенности;

Линии, образованные точками-гребнями, представляют собой линии водоразделов или непрерывные границы областей. Ясно, что основной задачей данного метода является именно поиск линий водоразделов.

Алгоритм, реализующий метод:

1. В местах локального минимума образуем «дырки», через которые вода начнет заполнять трехмерную поверхность;

2. Если вода с двух сторон гребня готова объединиться в один бассейн, устанавливаем перегородку;

3. Когда над водой останутся только загородки, останавливаем алгоритм.

Полученные таким образом перегородки и есть требуемые лини водоразделов.

При сегментировании вместе с пороговыми методами и методом водоразделов используются операции эрозии и дилатации.

Эрозия бинарного изображения A структурным элементом B обозначается $A \ominus B$ и задаётся выражением:

$$A \ominus B = \{z \in A | B_z \subseteq A\}.$$

При выполнении операции эрозии структурный элемент проходит по всем пикселам изображения. Если в некоторой позиции каждый единичный пиксел структурного элемента совпадет с единичным пикселом бинарного изображения, то выполняется логическое сложение центрального пиксела структурного элемента с соответствующим пикселом выходного изображения. В результате применения операции эрозии все объекты меньшие, чем структурный элемент, стираются, объекты, соединённые тонкими линиями становятся разъединёнными и размеры всех объектов уменьшаются.

Дилатация (наращивание) бинарного изображения A структурным элементом B обозначается $A \oplus B$ и задаётся выражением:

$$A \oplus B = \bigcup_{b \in B} A_b.$$

В данном выражении оператор объединения можно считать оператором, применяемым в окрестности пикселов. Структурный элемент B применяется ко всем пикселам бинарного изображения. Каждый раз, когда начало координат структурного элемента совмещается с единичным бинарным пикселом, ко всему структурному элементу применяется

перенос и последующее логическое сложение (логическое ИЛИ) с соответствующими пикселами бинарного изображения. Результаты логического сложения записываются в выходное бинарное изображение, которое изначально инициализируется нулевыми значениями.

Маркер – объект, который используется при реализации метода водораздела. Маркер ставится при ручном определении сегментируемых объектов. Он представляет собой связную компоненту, принадлежащую изображению. Будем различать внешние (соответствуют фону) и внутренние (относящиеся к объекту) маркеры. Процедура выбора маркера состоит из двух главных шагов:

- предобработка;
- выработка критериев, которым должны удовлетворять маркеры.

Маркеры могут иметь сложное описание, включающее размеры, форму, местоположение, расстояния, текстурные и другие признаки. Самым большим достоинством маркеров является то, что можно использовать априорные знания о задаче и эффективно использовать их при решении.

Функция morphologyEx() может выполнять расширенные морфологические преобразования, используя в качестве основных операций эрозию и расширение.

```
void cv::morphologyEx (InputArray src, OutputArray dst, int op,
InputArray kernel, Point anchor = Point(-1,-1), int iterations = 1, int
borderType = BORDER_CONSTANT, const Scalar
borderValue = morphologyDefaultBorderValue() )
```

src	Исходное изображение.
dst	Целевое изображение того же размера и типа, что и исходное изображение.
op	Тип морфологической операции
kernel	Структурирующий элемент. Его можно создать с помощью getStructuringElement .
anchor	Положение якоря с ядром. Отрицательные значения означают,

что якорь находится в центре ядра.

iterations	Количество применений эрозии и дилатации.
borderType	Метод экстраполяции пикселей.
borderValue	Значение границы в случае постоянной границы.

Функция dilate() расширяет исходное изображение с помощью указанного элемента структурирования, который определяет форму окрестности пикселя, по которой берется максимум:

```
void cv::dilate(InputArray src, OutputArray dst, InputArray kernel, Point
anchor = Point(-1,-1), int iterations = 1, int borderType =
BORDER_CONSTANT, const Scalar borderValue=
morphologyDefaultBorderValue() )
```

Ниже представлена программа, которая работает в двух режимах, автоматическом и ручном. Авто режим позволяет без участия человека разбить изображение со статичной картинки и видеофайла. Применен следующий алгоритм:

- Получение изображения;
- Перевод в бинарный вид и получение массива контуров;
- Создание маски с отрисованными контурами;
- Применение watershed к этой маске, предварительно задав цвет;
- Вывод итогового изображения.

Посредством изменения параметра thresh в функции threshold можно убрать или добавить более мелкие отслеживаемые области. Для лучшего разбиения изображения желательно после определения контуров исключить те, длина которых слишком мала (<20 пикселей). Это также поможет исключить излишнюю сегментацию. Для улучшения работы алгоритма можно использовать метод нахождения локальных минимумов/максимумов.

В режиме ручного управления происходит несколько другой алгоритм подготовки изображения. Написана ручная маркировка для статичного изображения. Алгоритм работы:

- Получение изображения;
- Создание маски путем ручного выделения объектов;
- Получение контура с маски, а не с изображения;
- Нанесение на новое изображение найденные контуры и применение к ним watershed;
- Вывод итогового изображения.


```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import matplotlib
matplotlib.use('Qt5Agg')
import numpy
from scipy.ndimage import label


img = cv2.imread('lab6_1.jpg')
img = cv2.resize(img, (0, 0), fx=0.5, fy=0.5)

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

#thresh1 = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, numpy.ones((3, 3),
dtype=int),iterations=2)
#thresh1 = cv2.dilate(thresh1,numpy.ones((3, 3), dtype=int),iterations=3)

kernel = np.ones((3,3),dtype=int)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
sure_bg = cv2.dilate(opening,kernel,iterations=3)
```

```
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)

sure_fg = np.uint8(sure_fg)
thresh1 = cv2.subtract(sure_bg,sure_fg)

# ret, markers = cv2.connectedComponents(sure_fg)
# markers = markers+1
# markers[thresh1==255] = 0
# markers = cv2.watershed(img,markers)
# img[markers == -1] = [255,0,0]


img = cv2.resize(img, (0, 0), fx=0.5, fy=0.5)
thresh = cv2.resize(thresh, (0, 0), fx=0.5, fy=0.5)
thresh1 = cv2.resize(thresh1, (0, 0), fx=0.5, fy=0.5)
cv2.imshow("IMG", img)
cv2.imshow("V1", thresh)
cv2.imshow("V1+", thresh1)
cv2.moveWindow("IMG", 0, 290)
cv2.moveWindow("V1", 320, 290)
cv2.moveWindow("V1+", 320, 580)

# v2
def segment_on_dt(a, img):
    border = cv2.dilate(img, None, iterations=5)
    border = border - cv2.erode(border, None)

    dt = cv2.distanceTransform(img, 2, 3)
    dt = ((dt - dt.min()) / (dt.max() - dt.min()) * 255).astype(numpy.uint8)
    _, dt = cv2.threshold(dt, 180, 255, cv2.THRESH_BINARY)
    lbl, ncc = label(dt)
    lbl = lbl * (255 / (ncc + 1))
    # Completing the markers now.
    lbl[border == 255] = 255

    lbl = lbl.astype(numpy.int32)
    cv2.watershed(a, lbl)

    lbl[lbl == -1] = 0
    lbl = lbl.astype(numpy.uint8)
    return 255 - lbl


img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, img_bin = cv2.threshold(img_gray, 0, 255,cv2.THRESH_OTSU)
img_bin = cv2.morphologyEx(img_bin, cv2.MORPH_OPEN,numpy.ones((3, 3), dtype=int))

result = segment_on_dt(img, img_bin)
cv2.imshow("V2", result)
cv2.moveWindow("V2", 640, 290)

result[result != 255] = 0
result = cv2.dilate(result, None)
img[result == 255] = (0, 0, 255)
cv2.imshow("BORDERS", result)
cv2.moveWindow("BORDERS", 960, 290)


# k-means
```

```
img = cv2.imread('lab6_1.jpg')
img = cv2.resize(img, (0, 0), fx=0.25, fy=0.25)
Z = img.reshape((-1,3))

# convert to np.float32
Z = np.float32(Z)

# define criteria, number of clusters(K) and apply kmeans()
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 2
ret,label,center=cv2.kmeans(Z,K,None,criteria,10,cv2.KMEANS_RANDOM_CENTERS)

# Now convert back into uint8, and make original image
center = np.uint8(center)
res = center[label.flatten()]
res2 = res.reshape((img.shape))

cv2.imshow("KMeans", res2)
cv2.moveWindow("KMeans", 1280, 290)

cap1 = cv2.VideoCapture('v3.mp4')

while cap1.isOpened():
    ret1, frame1 = cap1.read()

    gray = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
    #thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, numpy.ones((3, 3),
dtype=int), iterations=2)
    #thresh = cv2.dilate(thresh, numpy.ones((3, 3), dtype=int), iterations=3)

    # render
    cv2.imshow('ORIGINAL', frame1)  # original
    cv2.imshow('WATERSHED', thresh)

    # move
    frame1 = cv2.resize(frame1, (0, 0), fx=0.5, fy=0.5)
    thresh = cv2.resize(thresh, (0, 0), fx=0.5, fy=0.5)
    gray = cv2.resize(gray, (0, 0), fx=0.5, fy=0.5)
    cv2.imshow("ORIGINAL", frame1)
    cv2.imshow("WATERSHED", thresh)
    cv2.imshow("GRAY", gray)
    cv2.moveWindow("ORIGINAL", 0, 0)
    cv2.moveWindow("WATERSHED", 320, 0)
    cv2.moveWindow("GRAY", 640, 0)

    # time.sleep(0.05)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap1.release()
cv2.destroyAllWindows()
```

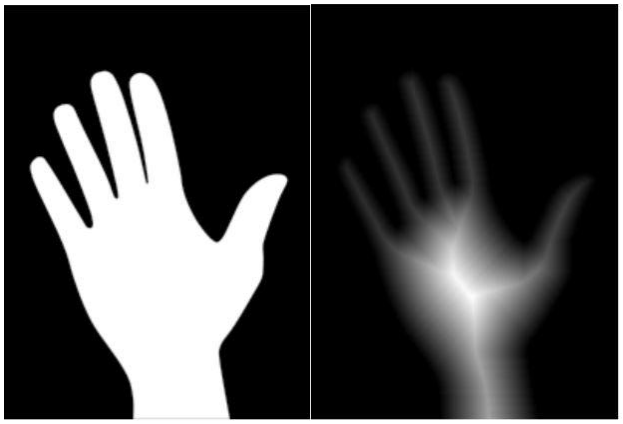
3.4. Отделение объектов от фона. Алгоритмы Distance Transform и Watershed

В предыдущем разделе рассмотрен метод морфологического водораздела. Он часто используется для отделения объектов на изображении от фона. Функция watershed(InputArray image, InputOutputArray markers) принимает изображение в градациях серого и изображения маркера. Маркеры являются изображением, где функции Watershed указываются объекты переднего плана и фон.

Ниже представлены: слева - входное изображение с маркерами, нарисованными вручную для того, чтобы отделить дорогу от всего остального; аправа - сегментированная дорога.

Distance Transform

Алгоритм Distance transform используется для автоматической генерации маркерного изображения для алгоритма Watershed. Результат работы алгоритма Distance transform представляет собой бинарное изображение, где значение каждого пикселя заменяется его расстоянием до ближайшего пикселя фона. Ниже представлены: входное изображение (слева) и результат алгоритма Distance transform (справа).



Программа, иллюстрирующая описанные функции, выполняет следующие действия: 1. Получение изображения (кадра из видео). 2. Преобразование изображения в бинарный вид. 3. Выполнение алгоритма Distance transform и нормализация результата в диапазоне [0,1]. 4. Пороговое преобразование для получения пиков и формирование маркеров для объектов переднего плана. 5. Поиск контуров. 6. Сегментация изображения.

Использованные функции

Функция	Описание
void cv::resize(InputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0, int interpolation=INTER_LINEAR)	Изменение размеров изображения
void cv::imshow(const String & winname, InputArray mat)	Отображение изображения в указанном окне
void cv::destroyWindow(const String & winname)	Уничтожение указанного окна
void cv::cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)	Преобразование изображения из одного цветового пространства в другое
double cv::threshold (InputArray src, OutputArray dst, double thresh, double maxval, int type)	Применение порога фиксированного уровня для каждого элемента массива.
void cv::distanceTransform (InputArray src, OutputArray dst, OutputArray labels, int distanceType, int maskSize, int labelType=DIST_LABEL_CCOMP)	Вычисление расстояния до ближайшего нулевого пикселя для каждого пикселя исходного изображения.
static Vec< _Tp, cn > cv::normalize (const Vec< _Tp, cn > &v)	Нормализации входного массива (используется для изменения диапазона значений интенсивности пикселей).
void cv::findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset = Point())	Поиск контуров в двоичном изображении

void drawContours(InputOutputArray image, InputArrayOfArrays contours, int contourIdx, const Scalar& color, int thickness=1, int lineType=8, InputArray hierarchy=noArray(), int maxLevel=INT_MAX, Point offset=Point())	Отрисовка контуров из заполненного массива
void cv::watershed(InputArray image, InputOutputArray markers)	Выполнение сегментации изображений на основе маркеров с использованием алгоритма водораздела
bitwise_and(image,image, mask= mask)	Производит поразрядное соединение двух массивов

Листинг программы C++

```
#include <iostream>
#include < opencv2/highgui/highgui.hpp>
#include < opencv2/imgproc/imgproc.hpp>
#include < opencv2/core/types.hpp>

using namespace std;
vector < cv::Vec3b> colors;

cv::Mat changeFrame(cv::Mat frame); // Функция преобразования изображения,
возвращает кадр

int main() {
cv::VideoCapture capture;
double k = 0.2;
capture.open( "hand.mp4" ); // Открываем файл

cv::Mat frame; //кадр
double fps = capture.get(CV_CAP_PROP_FPS); // Частота кадров

cv::Size size (k*capture.get(cv::CAP_PROP_FRAME_WIDTH),
k*capture.get(cv::CAP_PROP_FRAME_HEIGHT));
cv::VideoWriter writer("Result.avi" , CV_FOURCC('M','J','P','G'), fps, size);
while(1) {
capture >> frame; // Читаем кадр из файла
if(frame.empty()) break; // Если кадров больше нет - выходим
cv::resize(frame, frame, cv::Size(), k, k);
 cv::imshow( "Video Player", frame); // Выводим кадр

cv::Mat tmp;
 tmp = changeFrame(frame);//вызов функции преобразования изображения
writer.write(tmp); //Запись

cv::imshow( "Result", tmp);
char c = cv::waitKey(33); // Ждем 33 мс
if( c == 27 ) break; // Если нажали Esc-выходим
}

capture.release(); // Закрываем файл
writer.release();
cv::destroyWindow("Video Player"); // Закрываем окно
}

// Функция преобразования изображения
cv:: Mat changeFrame(cv::Mat frame) {
```

```
using namespace cv;

Mat src; src = frame;
 if (src.data == nullptr) return frame;

// Создаем бинарное изображение из исходного
Mat bw; cvtColor(src, bw, CV_BGR2GRAY);
threshold(bw, bw, 40, 255, CV_THRESH_BINARY);

// Выполняем алгоритм
 distance transform Mat dist;
distanceTransform(bw, dist, CV_DIST_L2, 3);

// Нормализуем изображение в диапазоне {0.0 1.0}
 normalize(dist, dist, 0, 1., NORM_MINMAX);
// Выполняем Threshold для определения пиков
// Это будут маркеры для объектов на переднем плане
threshold(dist, dist, .5, 1., CV_THRESH_BINARY);

// Создаем CV_8U версию distance изображения
// Это нужно для функции cv::findContours()
Mat dist_8u;
dist.convertTo(dist_8u, CV_8U);

// Находим все маркеры
vector<vector> contours;
findContours(dist_8u, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

auto ncomp = static_cast(contours.size());

// Создаем маркерное изображение для алгоритма watershed
Mat markers = Mat::zeros(dist.size(), CV_32SC1);

// Рисуем маркеры переднего плана
for (int i =0; i < ncomp; i++)
drawContours(markers, contours, i, Scalar::all(i+1), -1);

// Рисуем маркер фона, предполагая что изображение в центре
circle(markers, Point(5,5), 3, CV_RGB(255,255,255), -1);

// Выполняем алгоритм watershed
watershed(src, markers);

// Создаем результирующее изображение
Mat dst = Mat::zeros(markers.size(), CV_8UC3);

vector colors;
for(int i = 0; i < ncomp; i++){
 colors.emplace_back(frame.at(contours[i][0]));
}
// Выполняем помеченные объекты цветом
for (int i = 0; i < markers.rows; i++) {
       for (int j = 0; j < markers.cols; j++) {
            int index = markers.at(i,j);
            dst.at(i,j) = index > 0 && index <= ncomp ? colors.at(index-1) :
Vec3b(0,0,0);
} }

   return dst;
}
```

Представим другую программу (Python), ее алгоритм представлен ниже.

1) Получение изображения
2) В случае, если фон является белым, он заменяется на черный (т.к. фон будет помечаться белым маркером)
3) Улучшение качества изображения
4) Создание двоичного изображения из исходного
5) Применение алгоритма Distance Transform
6) Нормализация результата из предыдущего пункта в диапазоне от 0 до 1
7) Пороговое преобразование нормализованного изображения для получения пиков
8) Преобразование изображения в формат, необходимый для определения маркеров изображения
9) Создание маркерного изображения, маркер фона ставится в левом верхнем левом углу
10) Применение алгоритма Watershed
11) Создание итогового изображения, каждый распознанный объект раскрашивается в свой случайный цвет

Список использованных функция

Название функции	Описание
imread	Считывание изображения из файла
imshow	Вывод изображения
filter2D	Применяет к изображению переданный линейный фильтер
cvtColor	Переводить изображения из одного цветового пространства в другое
threshold	К каждому элементу изображения применяют порог фиксированного уровня
distanceTransform	Применение алгоритма Distance Transform
normalize	Нормализация значений
dilate	Расширение изображения по переданному структурирующему элементу
findContours	Нахождение контуров двоичного изображения
drawContours	Отрисовка контуров
circle	Отрисовка круга (в нашем случае маркер фона)
bitwise_not	
waitKey	Функция ожидания

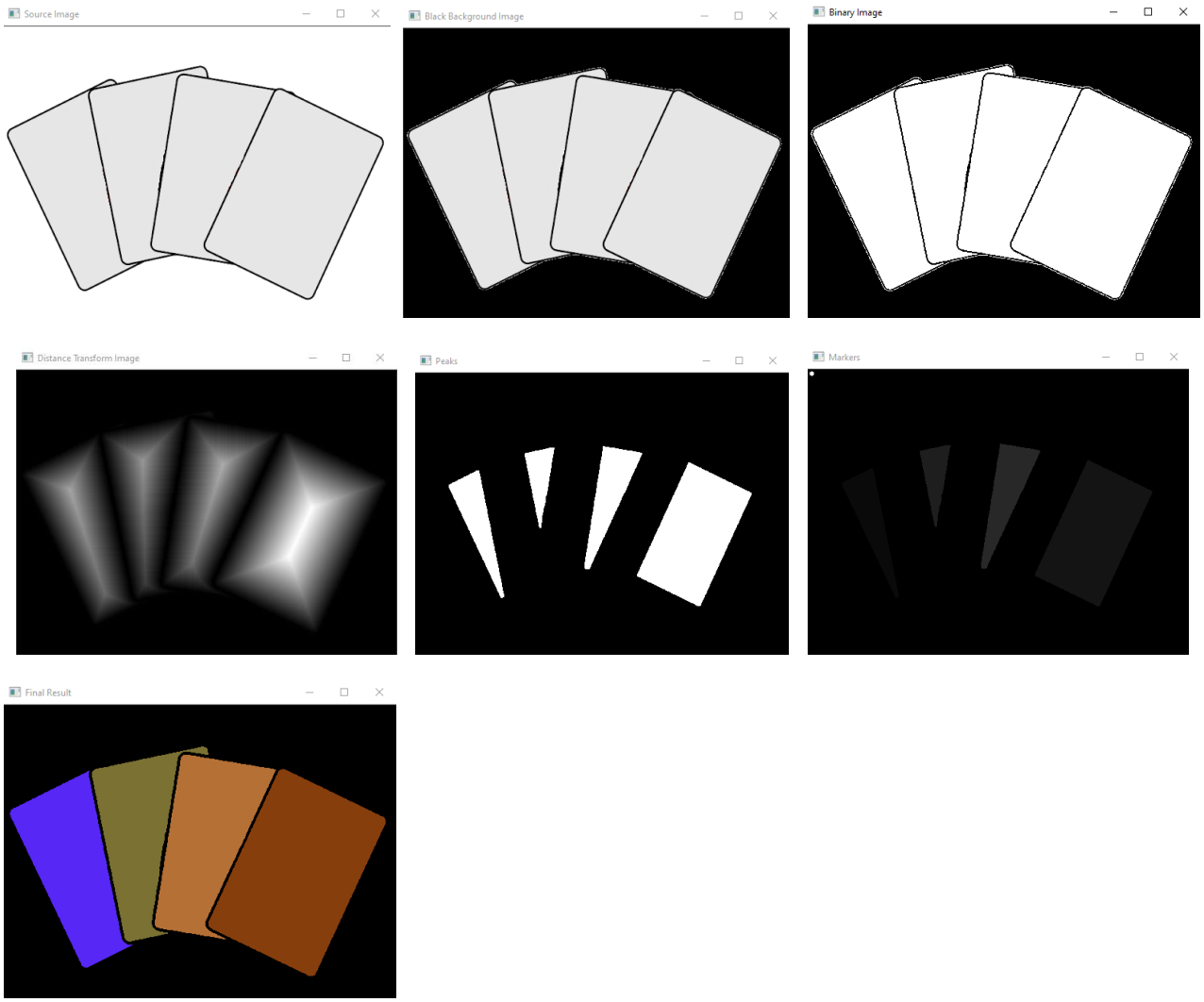
Листинг программы Python

```
import cv2
import numpy
```

```
import random

sourceImage = cv2.imread('source.jpeg')

# Вывод исходной картинки
cv2.imshow('Source Image', sourceImage)

# Изменения цвета фона на черный, если фон является белым
# т.к. маркер фона будет белого цвета
# и по другому этот маркер будет не распознать
sourceImage[numpy.all(sourceImage == 255, axis=2)] = 0

# Вывод картинки с черным фоном
cv2.imshow('Black Background Image', sourceImage)

# Улучшение качества изображения для увеличения резкости краев объектов
# Для этого применяется сильный фильтр с приближением по второй производной
# Результирующее изображения сохраняется в более широкий тип
customFilter = numpy.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]],
dtype=numpy.float32)
filterResult = cv2.filter2D(sourceImage, cv2.CV_32F, customFilter)
sharpenedImage = numpy.float32(sourceImage)
imgSharpenedResult = sharpenedImage - filterResult

# Преобразование улучшенного изображения в изображение в сером цвете
imgSharpenedResult = numpy.clip(imgSharpenedResult, 0, 255)
imgSharpenedResult = imgSharpenedResult.astype('uint8')
filterResult = numpy.clip(filterResult, 0, 255)
filterResult = numpy.uint8(filterResult)

# создание двочиного изображения из исходного
binaryImage = cv2.cvtColor(imgSharpenedResult, cv2.COLOR_BGR2GRAY)
_, binaryImage = cv2.threshold(binaryImage, 40, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)
cv2.imshow('Binary Image', binaryImage)

# Применение алгоритма Distance Transform
distanceTransformImage = cv2.distanceTransform(binaryImage, cv2.DIST_L2, 3)

# Нормализация результата в диапозоне [0, 1]
# для возможности различать пороги изображения
cv2.normalize(distanceTransformImage, distanceTransformImage, 0, 1.0,
cv2.NORM_MINMAX)

# Вывод нормализованного алгоритма после отработки алгоритма Distance Transform
cv2.imshow('Distance Transform Image', distanceTransformImage)

# Пороговое преобразование для получения пиков
# эти пики и будут маркерами определенных объектов
_, distanceTransformImage = cv2.threshold(distanceTransformImage, 0.4, 1.0,
cv2.THRESH_BINARY)

# Небольшое увеличение изображения
scalingFilter = numpy.ones((3, 3), dtype=numpy.uint8)
distanceTransformImage = cv2.dilate(distanceTransformImage, scalingFilter)

# Вывод пиков изображения
cv2.imshow('Peaks', distanceTransformImage)

# Создание изображения в формате CV_8U, с которым может работать
```

```
# алгоритм определения контуров объектов
distanceTransformImage_8u = distanceTransformImage.astype('uint8')

# Нахождение всех маркеров
edges, _ = cv2.findContours(distanceTransformImage_8u, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Создание маркерного изображения для применения алгоритма Watershed
objectMarkers = numpy.zeros(distanceTransformImage.shape, dtype=numpy.int32)

# Отрисовка маркеров предметов
for i in range(len(edges)):
    cv2.drawContours(objectMarkers, edges, i, (i + 1), -1)

# Установка маркера фона (левый верхний угол)
cv2.circle(objectMarkers, (5, 5), 3, (255, 255, 255), -1)
object_markers_8u = (objectMarkers * 10).astype('uint8')
cv2.imshow('Markers', object_markers_8u)

# Применение алгоритма Watershed
cv2.watershed(imgSharpenedResult, objectMarkers)
marker = objectMarkers.astype('uint8')
marker = cv2.bitwise_not(marker)


# Генерация различных цветов для различных предметов
objectColors = []
for edge in edges:
    objectColors.append((random.randint(0, 256), random.randint(0, 256),
random.randint(0, 256)))

# Создание итогового изображения
result = numpy.zeros((objectMarkers.shape[0], objectMarkers.shape[1], 3),
dtype=numpy.uint8)

# Закрашивание предметов соответствующими цветами
for i in range(objectMarkers.shape[0]):
    for j in range(objectMarkers.shape[1]):
        index = objectMarkers[i, j]
        if index > 0 and index <= len(edges):
            result[i, j, :] = objectColors[index - 1]

# Вывод результата
cv2.imshow('Final Result', result)

cv2.waitKey()
```

Скриншоты иллюстрируют каждый шаг выполнения программы, и итоговое изображение, на котором каждый распознанный объект раскрашивается в какой-то свой случайный цвет



3.5. Сегментация с использованием функции Canny

Canny Edge Detection – это популярный алгоритм обнаружения контуров. Он был разработан Джоном Ф. Кэнни в 1986 году. Это многоступенчатый алгоритм, состоящий из следующих этапов:

1. Подавление шума.
   Обнаружение контуров чувствительно к шуму, поэтому первым делом удаляем шум в изображении, применяя гауссовский фильтр 5*5.
2. Нахождение градиента яркости изображения.
   Нахождение градиента контура и направления для каждого пикселя.
3. Немаксимальное подавление.
   На каждом пикселе проверяется, является ли пиксель локальным максимумом в его окрестности в направлении градиента.
4. Пороговое значение гистерезиса.

На этом этапе решается, какие из ребер действительно являются ребрами, а какие нет. Для этого используются два пороговых значения.

Основная функция библиотеки OpenCV:

cv2.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]])

image – входное изображение,
threshold1 – минимальное пороговое значение,
threshold2 – максимальное пороговое значение,
edges – выходная карта контуров (может совпадать с image),
apertureSize – размер апертуры для оператора Sobel(),
L2gradient – флаг, указывающий, следует ли использовать более точную норму для вычисления градиента изображения.

Алгоритм программы включает следующие этапы:

1. Подключение библиотек OpenCV и matplotlib (для вывода изображений).
2. Вывод на экран исходного изображения
3. Сегментация изображения
    a. Конвертация изображения из цветовой палитры RGB в цветовую палитру HSV
    b. Обозначение границ цвета (светлая и темная)
    c. Создание маски изображения с использованием граничных цветов
    d. Применение маски поверх исходного изображения
    e. Вывод на экран замаскированного изображения
4. Обнаружение контуров на изображении
    a. Конвертация изображения из цветовой палитры RGB в черно-белую палитру и применение гауссова размытия (для лучшего обнаружения контуров)
    b. Применение порога фиксированного уровня к каждому пикселю для уменьшения шума

c. Выделение контуров с помощью функции findContours
d. Отображение контуров поверх исходного изображения с помощью функции drawContours
e. Вывод на экран исходного изображения с наложенными поверх контурами

5. Применение функции Canny к исходному изображению
6. Вывод на экран контуров, найденных с помощью функции Canny

Список использованных функций OpenCV

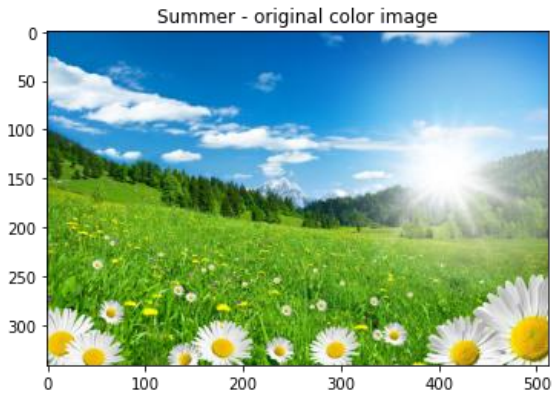
Название функции	Описание
imread	Загрузка изображения из файла
cvtColor	Конвертация изображения из одной цветовой палитры в другую
inRange	Проверка, лежат ли элементы массива между элементами двух других массивов
bitwise_and	Операция побитового И над пикселями изображения
GaussianBlur	Размытие изображения с помощью фильтра Гаусса
threshold	Применение порога фиксированного уровня к каждому элементу массива
findContours	Выделение контуров
drawContours	Отображение контуров или контуров с заливкой
Canny	Детектор ребер

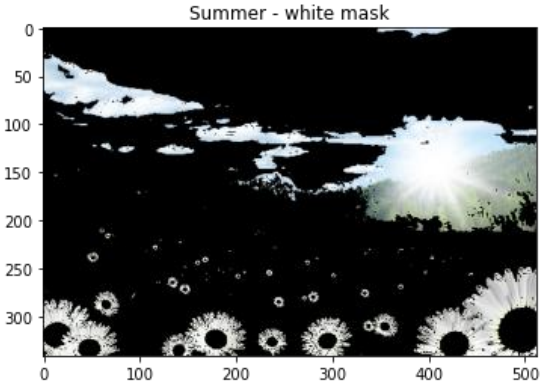
Листинг программы

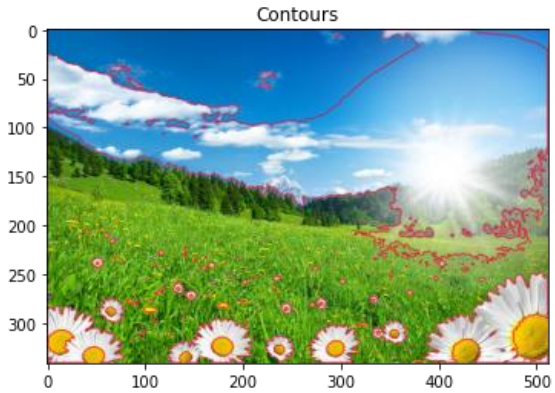
```
import cv2
from matplotlib import pyplot as plt
import numpy as np
img = cv2.imread("summer.jpg")
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title("Summer - original color image")

# Сегментация изображения
hsv_image = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)

light_white = (0, 0, 100)
dark_white = (145, 60, 255)

# Применение белой маски
mask_white = cv2.inRange(hsv_image, light_white, dark_white)
result = cv2.bitwise_and(img, img, mask=mask_white)

plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
plt.title("Summer - white mask")

# Обнаружение и отображение контуров
temp = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)   #Для лучшего обнаружения - ч/б и блюр
blurred = cv2.GaussianBlur(temp, (3, 3), 0)
T, binary_img = cv2.threshold(blurred, 155, 255, cv2.THRESH_BINARY)
```

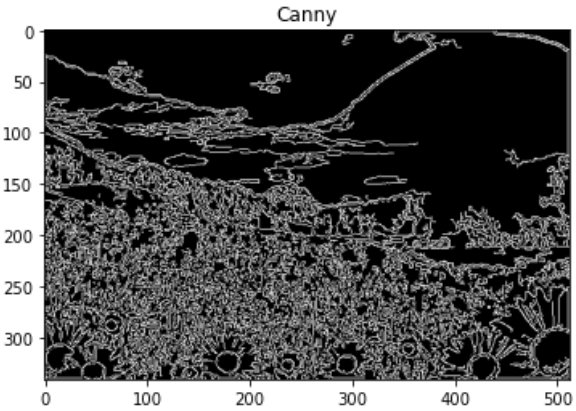
```
contours, hierarchy = cv2.findContours(binary_img.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

# Отображение контуров поверх изображения
cv2.drawContours(img, contours, -1, (60,20,220), 1, cv2.LINE_AA, hierarchy, 5)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Contours')

# Использование функции Canny
edges = cv2.Canny(img,1,300,L2gradient=False)
plt.imshow(cv2.cvtColor(edges, cv2.COLOR_BGR2RGB))
plt.title('Canny')
```

Скриншоты результатов работы программ









3.6. Алгоритмы сегментации, включая QuickShift

Сегментирование изображения – это разбиение его на области, содержащие пиксели, объединённые каким-либо общим свойством. Сегментация изображений до сих пор остаётся актуальной задачей из области технического зрения, поскольку используется при поиске и распознавании объектов. В качестве конкретных примеров можно привести использование в производстве для нахождения дефектов во

время сборки деталей, в медицине для обработки, например, рентгеновских снимков, в составлении карт местности по снимкам со спутников.

Результатом сегментации изображения является множество сегментов, которые вместе покрывают всё изображение, или множество контуров, выделенных из изображения. Все пиксели в сегменте похожи по некоторой характеристике или вычисленному свойству, например, по цвету, яркости или текстуре. Соседние сегменты значительно отличаются по этой характеристике.

Среди современных алгоритмов сегментации изображений выделяются пять алгоритмов: алгоритм Фельзенцваба, SLIC, QuickShift и стандартный и компактный алгоритмы сегментации по водоразделам (WaterShed).

1. Алгоритм Фельзенцваба: это сегментация на основе минимального остовного дерева. Позволяет разбить цифровое изображение на области точек с похожими свойствами, например, по однородности. Строится дерево, которое является подмножеством рёбер минимального веса графа без циклов. Рёбра рассматриваются в порядке возрастания веса, их пиксели включаются в области, если они не образуют цикла в графе и пиксели «подобны» существующим пикселям областей.

2. Алгоритм SLIC: расшифровывается как простая линейная итеративная кластеризация. Он не требует больших вычислительных мощностей. Алгоритм группирует пиксели в объединенном пятимерном пространстве цвета и плоскости изображения, чтобы создать эффективные и компактные, почти однородные суперпиксели.

3. Алгоритм QuickShift: этот метод сегментирует изображение на однородные области, применяя метод быстрого сдвига с начальными параметрами, а затем автоматически получает окончательное сегментированное изображение, изменяя значения параметров

быстрого сдвига. Этот способ подходит для изображений большого размера.

4. Алгоритм WaterShed: один из основных алгоритмов наращивания областей, рекурсивно выполняющих процедуру группировки пикселей в подобласти по заранее заданным критериям. Вместо использования цветного изображения в качестве входных данных этому методу требуется изображение с градиентом в градациях серого, где яркие пиксели обозначают границу между областями. Далее каждый отдельный бассейн формирует отдельный сегмент исходного изображения.

Блок-схема алгоритма QuickShift

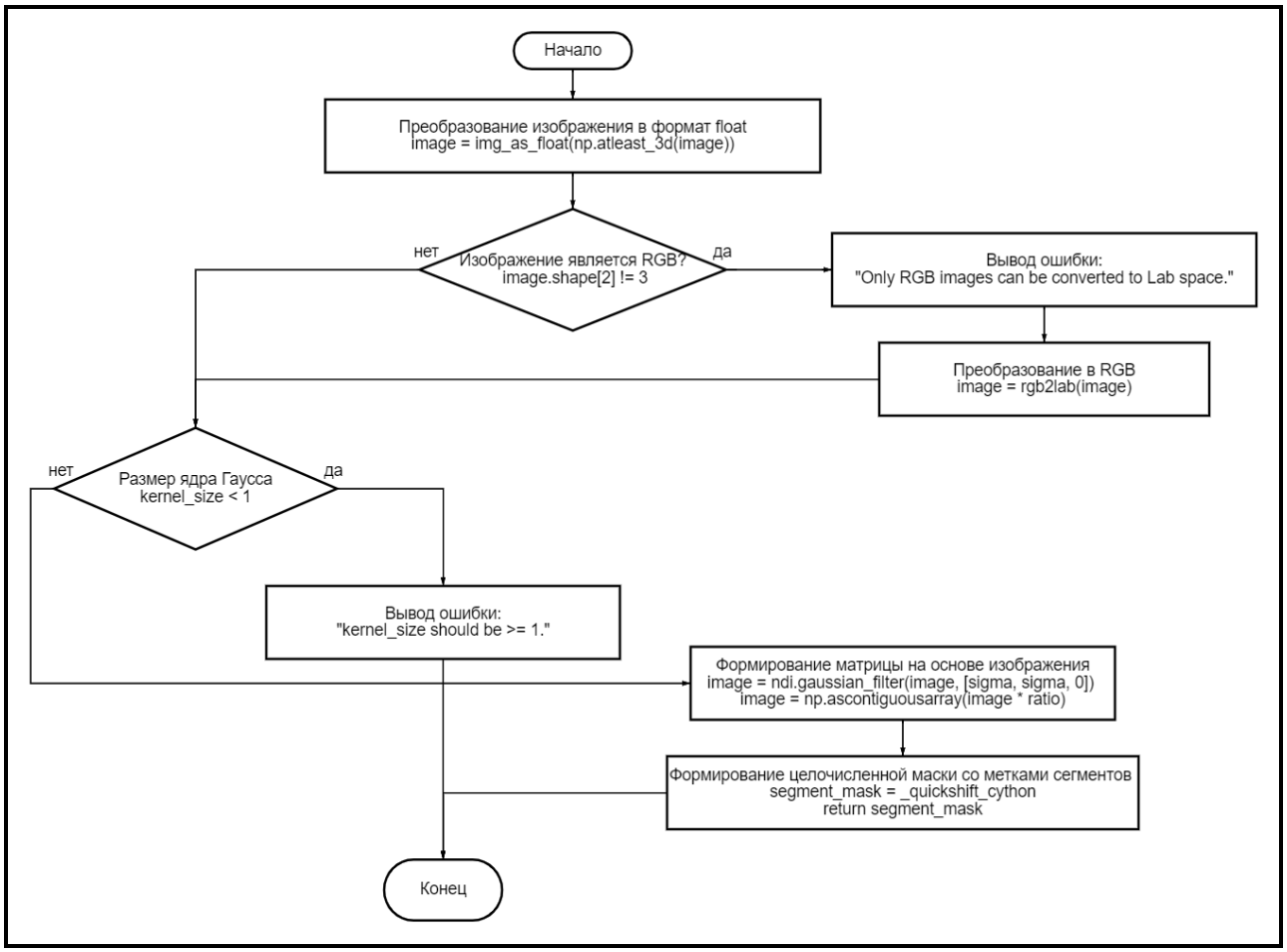


Таблица основных использованных функций

Функция	**Описание**

img_as_float (image[, force_copy])	преобразует изображение в формат с плавающей точкой со значениями, лежащими на промежутке [0, 1]
Felsenszwalb (image, scale=1, sigma=0.8, min_size =20, multichannel=True)	вычисляет эффективную сегментацию изображений на основе графов Фельзенцвальба.
slic(image, n_segments=100, compact ness=10.0, max_iter=10, sigma=0, spa cing=None, multichannel=True, convert2lab=None , enforce_connectivity=True, min_size _factor=0.5, max_size_factor=3, slic_zero=False, st art_label=None, mask=None)	сегментирует изображение с использованием кластеризации средних k-тых элементов в пространстве Color-(x,y,z)
quickshift(image, ratio=1.0, kernel_siz e=5, max_dist=10, return_tree=False, sigma=0, convert2lab=True, random_s eed=42)	сегментирует изображение с помощью кластеризации с быстрым сдвигом в пространстве Color-(x,y). А также дополнительно производит чрезмерную сегментацию изображения с помощью алгоритма быстрого переключения режимов (quickshift mode-seeking algorithm).
watershed(image, markers=None, con nectivity=1, offset=None, mask=None , compactness=0, watershed_line=Fals e)	выполняет алгоритм сегментации изображения по водоразделам, то есть начиная с определяемых пользователем маркеров, алгоритм водораздела обрабатывает значения пикселей как локальную топографию.
rgb2gray(rgb image)	вычисляет яркость RGB-изображения.
unique(ar, return_index=False, return_ inverse=False, return_counts=False, a xis=None)	находит уникальные элементы массива, а также возвращает их в отсортированном порядке.
matplotlib.pyplot.subplots(nrows=1, n cols=1, *, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gri dspec_kw=None, **fig_kw)	создайте фигуру и набор подзаголовков. Эта служебная оболочка позволяет за один вызов удобно создавать общие макеты подзаголовков, включая объект фигуры.

Функция ***quickshift()***

Входные параметры:
image : (width, height, channels) – входное изображение

ratio : float, optional – имеет значение между 0 и 1, уравновешивает близость цветового пространства и близость пространства изображения. Более высокие значения придают больший вес цветовому пространству.

kernel_size : float, optional - ширина ядра Гаусса, используемого для сглаживания плотности выборки. Чем больше значение, тем меньше кластеров.

max_dist : float, optional – точка отсечения данных. Чем больше значение, тем меньше кластеров.

return_tree : bool, optional – флаг для возврата полного дерева иерархии сегментации

sigma : float, optional – значение ширины сглаживания по Гауссу для предварительной обработки. Ноль означает отсутствие сглаживания.

convert2lab : bool, optional – флаг для преобразования входных данных в RGB

random_seed : int, optional – случайное значение сида для разрыва связей.

Возвращаемое значение функции ***quickshift***:
segment_mask : (width, height) – целочисленная маска, хранящая метки сегментов

Листинг программы

Код функции ascontiguousarray:

```
def ascontiguousarray(a, dtype=None, *, like=None):
    if like is not None:
        return _ascontiguousarray_with_like(a, dtype=dtype, like=like)

    return array(a, dtype, copy=False, order='C', ndmin=1)
```

Код функции gaussian_filter:

```
def gaussian_filter(input, sigma, order=0, output=None,
                    mode="reflect", cval=0.0, truncate=4.0):
    input = numpy.asarray(input)
    output = _ni_support._get_output(output, input)
    orders = _ni_support._normalize_sequence(order, input.ndim)
    sigmas = _ni_support._normalize_sequence(sigma, input.ndim)
    modes = _ni_support._normalize_sequence(mode, input.ndim)
    axes = list(range(input.ndim))
    axes = [(axes[ii], sigmas[ii], orders[ii], modes[ii])
            for ii in range(len(axes)) if sigmas[ii] > 1e-15]
    if len(axes) > 0:
        for axis, sigma, order, mode in axes:
            gaussian_filter1d(input, sigma, axis, order, output,
                              mode, cval, truncate)
            input = output
    else:
        output[...] = input[...]
    return output
```

Файл _quickshift.py:

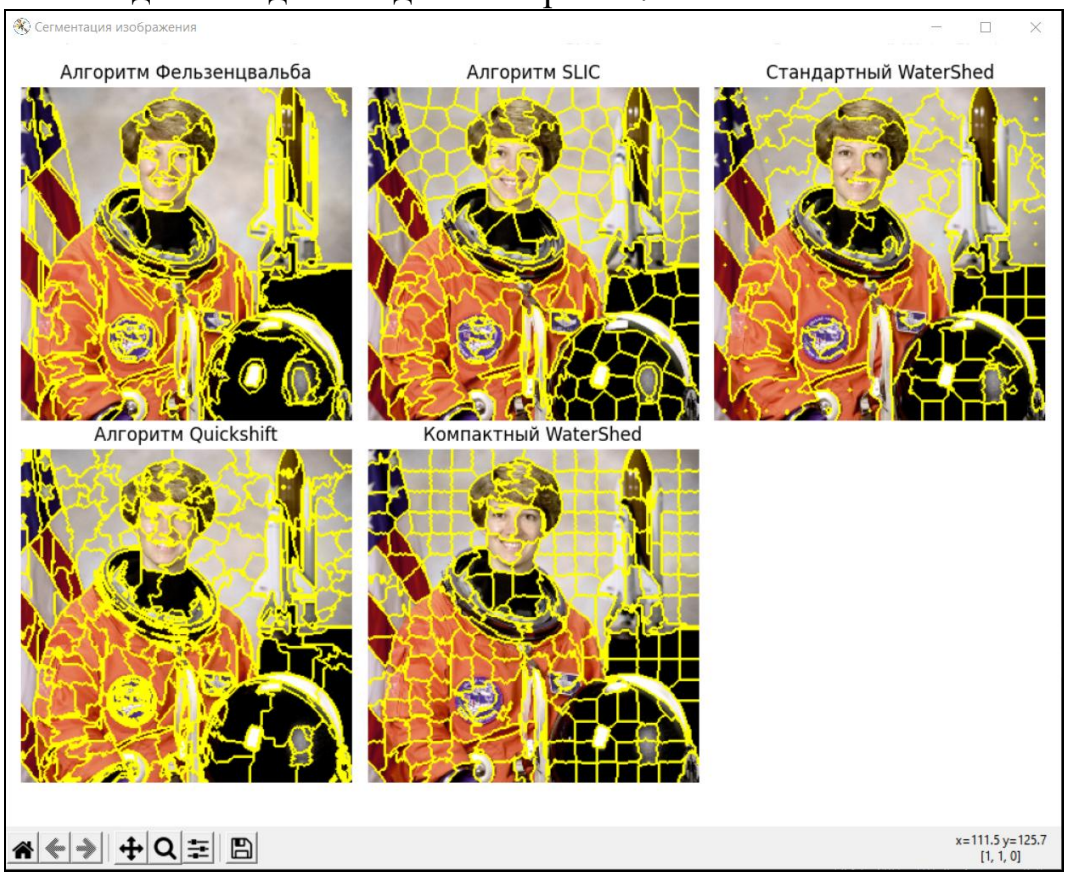
```
import numpy as np

import scipy.ndimage as ndi

from ..util import img_as_float
from ..color import rgb2lab

from ._quickshift_cy import _quickshift_cython


def quickshift(image, ratio=1.0, kernel_size=5, max_dist=10,
               return_tree=False, sigma=0, convert2lab=True, random_seed=42):
    image = img_as_float(np.atleast_3d(image))
    if convert2lab:
        if image.shape[2] != 3:
            ValueError("Only RGB images can be converted to Lab space.")
        image = rgb2lab(image)

    if kernel_size < 1:
        raise ValueError("`kernel_size` should be >= 1.")

    image = ndi.gaussian_filter(image, [sigma, sigma, 0])
    image = np.ascontiguousarray(image * ratio)

    segment_mask = _quickshift_cython(
        image, kernel_size=kernel_size, max_dist=max_dist,
        return_tree=return_tree, random_seed=random_seed)
    return segment_mask
```

Файл main.py:

```
import matplotlib.pyplot as plt
import numpy as np

from skimage.segmentation import mark_boundaries
from skimage.data import astronaut
from skimage.util import img_as_float
from skimage.color import rgb2gray
from skimage.filters import sobel
from skimage.segmentation import felzenszwalb, slic, quickshift, watershed

img = img_as_float(astronaut()[::2, ::2])

segments_fz = felzenszwalb(img, scale=100, sigma=0.5, min_size=50)
segments_slic = slic(img, n_segments=250, compactness=10, sigma=1, start_label=1)
segments_quick = quickshift(img, kernel_size=3, max_dist=6, ratio=0.5)
gradient = sobel(rgb2gray(img))
segments_standart_watershed = watershed(gradient, markers=250)
segments_compact_watershed = watershed(gradient, markers=250, compactness=0.005)

fzSegLength = len(np.unique(segments_fz))
slSegLength = len(np.unique(segments_slic))
qsSegLength = len(np.unique(segments_quick))
wcSegLength = len(np.unique(segments_compact_watershed))
wsSegLength = len(np.unique(segments_standart_watershed))

print(f"Количество сегментов (метод Фельзенцвальба): {fzSegLength}")
print(f"Количество сегментов (метод SLIC): {slSegLength}")
print(f"Количество сегментов (метод QuickShift): {qsSegLength}")
print(f"Количество сегментов (стандартный алгоритм сегментации по водоразделам):
```

```
{wsSegLength}")
print(f"Количество сегментов (компактный алгоритм сегментации по водоразделам):
{wcSegLength}")

fig, ax = plt.subplots(2, 3, figsize=(10, 10), sharex=True, sharey=True)
fig.canvas.set_window_title('Сегментация изображения')

ax[0, 0].imshow(mark_boundaries(img, segments_fz))
ax[0, 0].set_title("Алгоритм Фельзенцвальба")
ax[0, 1].imshow(mark_boundaries(img, segments_slic))
ax[0, 1].set_title('Алгоритм SLIC')
ax[1, 0].imshow(mark_boundaries(img, segments_quick))
ax[1, 0].set_title('Алгоритм Quickshift')
ax[1, 1].imshow(mark_boundaries(img, segments_compact_watershed))
ax[1, 1].set_title('Компактный WaterShed')
ax[0, 2].imshow(mark_boundaries(img, segments_standart_watershed))
ax[0, 2].set_title('Стандартный WaterShed')

for a in ax.ravel():
    a.set_axis_off()

plt.tight_layout()
plt.show()
```

Сегментное выделение для каждого алгоритма:



Количество сегментных выделений каждого алгоритма:

```
C:\Users\Антон\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Антон/PycharmProjects/pythonProject/main.py
Количество сегментов (метод Фельзенцвальба): 194
Количество сегментов (метод SLIC): 196
Количество сегментов (метод QuickShift): 695
Количество сегментов (стандартный алгоритм сегментации по водоразделам): 256
Количество сегментов (компактный алгоритм сегментации по водоразделам): 256
```

По результатам работы программы видно, что наилучшие результаты показывает алгоритм QuickShift. Данный алгоритм выделяет наибольшее количество сегментов изображения. Таким образом, можно сделать вывод, что QuickShift является наиболее эффективным методом сегментации изображения при поиске и распознавании объектов.

3.7. Алгоритм сегментации MeanShift

Алгоритм MeanShift объединяет сущности со схожими признаками в один сегмент. Так пиксели с близкими признаками попадают в один сегмент, образуя новое изображение с однородными областями

Программа выполняет следующие действия: Подключение библиотеки OpenCV; Считывание исходного изображения; Задание данных для сегментации; Сегментирование изображения; Вывод изображения на экран.

Список использованных функций

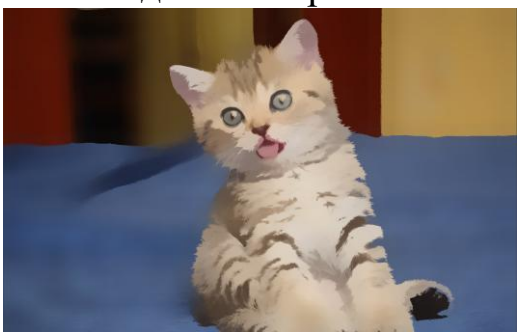
Название функции	Описание
imread	Считывание исходного изображения
pyrMeanShiftFiltering	Выполняет начальный шаг сегментации Mean Shift
imShow	Показ сегментированного изображения
waitKey	Ожидание нажатия клавиши от пользователя

```
import cv2

image = cv2.imread("test.jpg", 1);

spatialRadius = 35; // Пространственный радиус
colorRadius = 60; // Цветовой радиус
pyramidLevels = 3; // Гауссова пирамида будет иметь pyramidLevels+1 уровней
imageSegment = cv2.pyrMeanShiftFiltering(image, spatialRadius, colorRadius,
pyramidLevels);

cv2.imshow("MeanShift", imageSegment);
cv2.waitKey(0);
```

Результат работы программы:

Исходное изображение:

Выходное изображение:



3.8. Сегментация с помощью нахождения краев и контуров

Одним из инструментов сегментации являются функции выделения краев и контуров. Контур объекта — это кривая, соединяющая точки вдоль объекта. Кривая отделяет объект от фона.

Все вместе взятые методы для работы с контурами называют **контурным анализом**. Благодаря существованию такого анализа можно рассматривать только контуры объекта и тем самым можно отойти от пространства изображения к пространству контуров. Такой переход понижает трудоемкость алгоритмов и вычислений, связанных с ними.

Для поиска контуров в OpenCV необходимо сначала перевести изображение в черно-белую палитру и представить его в бинарном формате. Затем используется сама функция нахождения контуров findContours(image, режим нахождения, метод упаковки). Где image – это изображение, на котором будет происходить поиск контуров. Режим

нахождения – это то, как будут сгруппированы найденные контуры. Существует 4 режима:

1. RETR_LIST – все контуры будут не сгруппированы
2. RETR_TREE – размещает контуры в иерархию вложенных контуров
3. RETR_EXTERNAL – демонстрирует только крайние внешние контуры
4. RETR_CCOMP – размещает контуры в 2-х уровневую иерархию

Метод упаковки. Существует 4 метода упаковки контуров:

1. CHAIN_APPROX_NONE – контуры хранятся в виде отрезков
2. CHAIN_APPROX_SIMPLE – контуры будут склеены между собой
3. CHAIN_APPROX_TC89_L1 и CHAIN_APPROX_TC89_KCOS – к этим методом упаковки применяется алгоритм аппроксимации Tec-Chin

Найденные контуры накладываются поверх исходного изображения с помощью функции drawContours(image, контур1, индекс, цвет, толщина, тип линии, контур2, слой).

- **Image -** это изображение, на которое будут наложены найденные контуры.
- **Контур1 и контур2** – контуры найденные функцией **findContours().**
- **Индекс –** индекс, отображения контуров (-1 если нужно отобразить все контуры)
- **Цвет –** цвет контура
- **Толщина –** толщина контура
- **Тип линии –** тип соединения точек вектора (LINE_AA **– сглаженная линия,** LINE_4 **– четырёх связанная линия,** LINE_8 **– восьми связанная линия)**
- **Слой** – это индекс слоя, который требуется отобразить (1 если требуется отобразить все контуры)

В зависимости от цели нахождения и отображения краев и контуров, выбираются требуемые для этого режимы нахождения, методы упаковки и типы линий, а также остальные необходимые параметры. Поэтому для

одного изображения существуют различные варианты нахождения контуров.

В представленной ниже программе будет продемонстрировано два варианта нахождения контуров и их отображения на исходном изображении, благодаря изменениям в типе линий соединений, режиме нахождения контуров и их упаковки. А также сегментация изображения с помощью наложения маски и использования функции Canny.

Использованные функции:

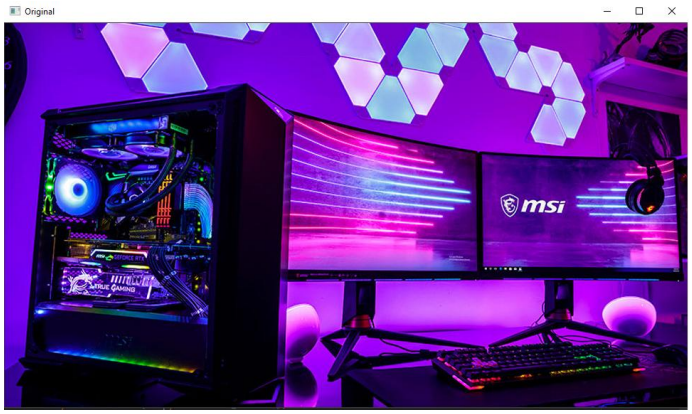
cv2.imread	Импортирование изображение из указанного файла
cv2.imshow	Используется для отображения изображения в новом окне
cv2.cvtColor	Изображение конвертируется из одной палитры в другую
cv2.inRange	Проверка порога изображения
cv2.bitwise_and	Побитовое И между пикселями изображения
cv2.findContours	Функция нахождения контуров
cv2.drawContours	Функция отображения найденных контуров
cv2.threshold	К каждому пикселю изображение применяется порог фиксированного уровня
cv2.Canny	Функция нахождения контуров и ребер
cv2.waitKey	Ожидание нажатия клавиши

Листинг программы

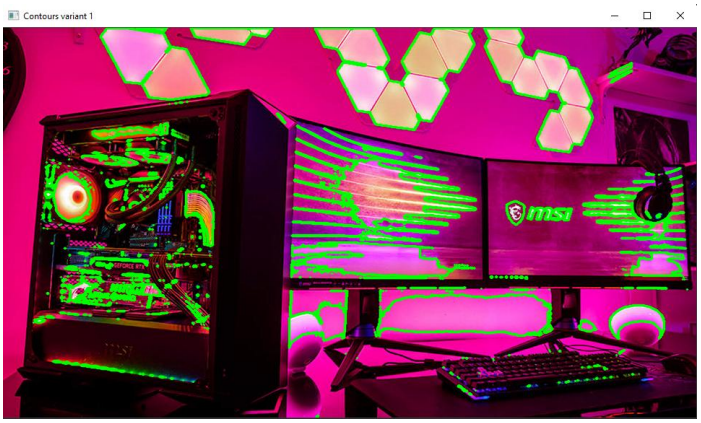
```
import cv2
# Изначальное изображение
img = cv2.imread('summer1.jpg')
cv2.imshow("Original", img)
# Поиск контуров и их отображение поверх начального изображении(Вариант 1)
newImg1 = cv2.cvtColor(img.copy(), cv2.COLOR_RGB2GRAY)
# Переводим изображение в бинарный вид
ret1, tresh1 = cv2.threshold(newImg1, 127, 255, cv2.THRESH_BINARY)
# Находим контуры и затем отображаем их
# Найденные контуры будут храниться в виде иерархии контуров и будут сгруппированы
contr1, contr2 = cv2.findContours(tresh1.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
# Контуры будут соединены сглаженной линией
cv2.drawContours(img, contr1, -1, (0, 255, 0), 2, cv2.LINE_AA, contr2, 1)
cv2.imshow("Contours version 1", cv2.cvtColor(img.copy(), cv2.COLOR_BGR2RGB))
# Поиск контуров и их отображение поверх начального изображении(Вариант 2)
newImg2 = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
ret2, tresh2 = cv2.threshold(newImg2, 20, 255, cv2.THRESH_BINARY)
# Найденные контуры будут храниться в виде отрезков и не будут сгруппированы
contr3, contr4 = cv2.findContours(tresh2.copy(), cv2.RETR_LIST,
```

```
cv2.CHAIN_APPROX_NONE)
cv2.drawContours(img, contr3, -1, (0, 255, 0), 1, cv2.LINE_4, contr4, 1)
cv2.imshow("Contours version 2", cv2.cvtColor(img.copy(), cv2.COLOR_BGR2RGB))
# Сегментация изображения с применением маски заданного цвета
img1 = cv2.cvtColor(img.copy(), cv2.COLOR_RGB2HSV)
bot = (0, 0, 100)
top = (200, 200, 255)
# Создаем маску желаемого  цвета
mask = cv2.inRange(img1, bot, top)
# Применяем созданную маску
temp = cv2.bitwise_and(img, img, mask=mask)
cv2.imshow("Segmentation ", cv2.cvtColor(temp, cv2.COLOR_BGR2RGB))
# Функция Canny
canny = cv2.Canny(img, 1, 100, L2gradient=True)
cv2.imshow("Canny", cv2.cvtColor(canny, cv2.COLOR_BGR2RGB))

# для завершения демонстрации работы программы
if cv2.waitKey(0) & 0xff == 27:
cv2.destroyAllWindows()
```

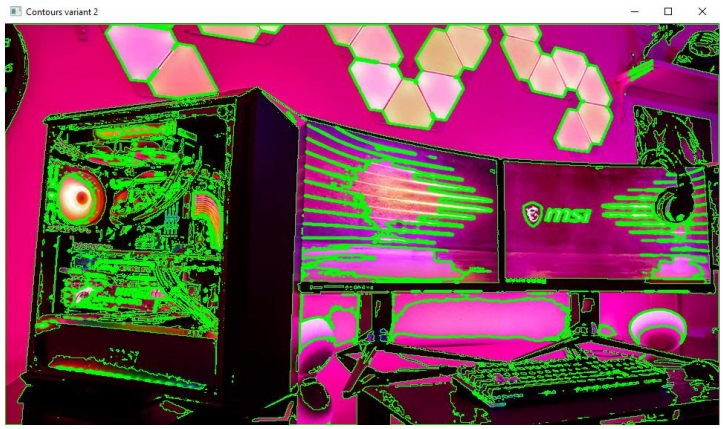
Original image



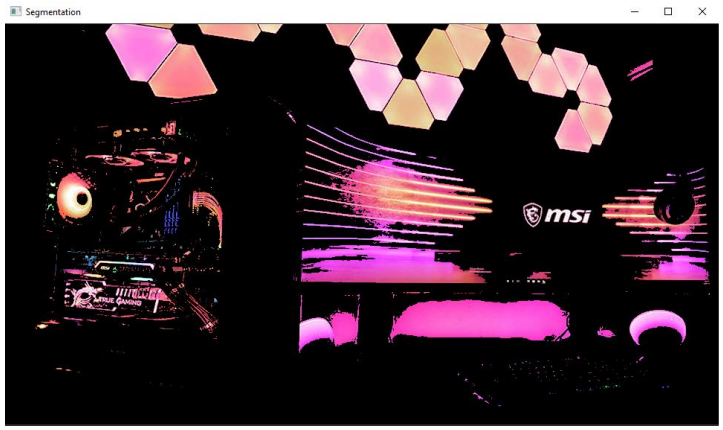
Contours



Contours version 2



Segmentation



3.9. Суперпиксельная сегментация

Одной из наиболее часто решаемых задач при обработке изображений является сегментация – разбиение изображения на непересекающиеся области, покрывающие все изображение и однородные по некоторым признакам: цвет, яркость, текстура и т. д. Как известно, большинство стандартных алгоритмов обработки изображений используют их представление в виде регулярной пиксельной сетки, что обусловлено, в большей степени, исходным способом хранения изображений в цифровой форме. Однако это не всегда оптимально с многих точек зрения и, прежде всего, для организации последующей обработки. В этом плане относительно новым и активно развиваемым, прежде всего, за рубежом, является подход, основанный на так называемой суперпиксельной сегментации (СС). СС реализует разбиение изображения на множество

мелких фрагментов (суперпикселей), представляющих из себя относительно однородные группы расположенных рядом пикселей. Каждый суперпиксель потенциально является атомарным регионом (фрагментом) изображения, т. е. все входящие в него пиксели рассматриваются при дальнейшей обработке как единое целое. При этом суперпиксели не обязательно должны иметь правильную форму и, естественно, всегда имеется определенное число ошибок, допускаемых при стремлении разбить изображение на однородные фрагменты [38].

Использование суперпикселей обусловлено несколькими причинами:

• Они несут больше информации, чем пиксели.

• Суперпиксели имеют перцепционное значение, поскольку пиксели, принадлежащие данному суперпикселю, обладают схожими визуальными свойствами.

• Они обеспечивают удобное и компактное представление изображений, которое может быть очень полезно для задач, требующих большого объема вычислений.

Суперпиксельное разбиение является результатом работы особых алгоритмов разбиения изображения. Самыми примечательными из них являются: SLIC (простая линейная итеративная кластеризация), SEEDS (суперпиксели извлечённые с помощью энергетического отбора проб сидов), LSC (линейная спектральная сегментация)

Простая линейная итеративная кластеризация (SLIC) Этот алгоритм генерирует суперпиксели путем кластеризации пикселей на основе их цветового сходства и близости в плоскости изображения. Это делается в пятимерном пространстве [labxy], где [lab] - это цветовой вектор пикселя в цветовом пространстве CIELAB, а xy - позиция пикселя. Изначально создаётся N центров кластеров. Если пиксель попадает в

область поиска кластера, то он проверяется на соответствие с центральным пикселем кластера, и если он попадает, то в данном кластере через средние находится новый центр. Сам пиксель в свою очередь будет привязан к кластеру.

Суперпиксели извлечённые с помощью энергетического отбора проб сидов (SEEDS) Алгоритм начинается с полного(всё изображение делиться на фиксированное число суперпикчелей) суперпиксельного разбиения, которое итеративно уточняется. Уточнение выполняется перемещением границ суперпикселей или, что то же самое, путем обмена пикселями между соседними суперпикселями. Используется специальная энергетическая функция, которая может быть эффективно максимизирована и позволяет обеспечить однородность цветового распределения, а также стремиться к обеспечению плавных форм границ.

Линейная спектральная сегментация(LSC) В спектральной кластеризации сродство, а не абсолютное местоположение (то есть kозначает), определяет, какие точки попадают под какой кластер. Последнее особенно полезно при решении проблем, когда данные образуют сложные формы. В LSC мы сопоставляем каждый пиксель изображения с точкой в десятимерном пространстве. Пространство признаков, в котором для сегментации применяется метод взвешенных K-средних (разбиения элементов на K кластеров в зависимости от расстояний до их центров). Алгоритм K-средних стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров.

Для исследования работы описанных выше алгоритмов написана программа, которая включает следующие этапы:

1) Поиск и запись имён фалов из директивы images в список имён

файлов;

2) Вывод данного списка изображений;

3) Ввод пользователем имени изображения, если имя файла неверно или файлов в директории images нет, выход из программы;

4) Вывод изображения на экран;

5) Применение к изображению Гаусовского размытия и вывод результата на экран;

6) Создание суперпикселей по алгоритмам SLIS,SEEDS и LSC;

7) Получение контурной маски и применение её к изображению для всех 3 алгоритмов;

8) Вывод полученных сегментированных изображений на экран и запись результатов в папку out;

9) Ожидание нажатия клавиши;

Использованные функции

cv2.imread()	Считывает изображение
cv2.imshow()	Выводит изображение в отдельном окне
cv2.GaussianBlur()	Размывает изображение для эффективной сегментации
iterate()	Производим итерации полученных наборов суперпикселей
getLabelContourMask()	Возвращает маску контуров суперпикселей изображения
cv2.bitwise_not()	Операция поразрядного “НЕ”. Используется для получения видимой маски контуров, которая накладывается на изображение
cv2.bitwise_and()	Операция поразрядного “И”. Используется для совмещения изображения и маски контуров суперпикселей
cv2.ximgproc.createSuperpixelSEEDS()	Создание суперпикселей по алгоритму SEEDS
cv2.ximgproc.createSuperpixelSLIC()	Создание суперпикселей по алгоритму SLIC. Вид используемого алгоритма задается в аргументах функции
createSuperpixelLSC	Реализует алгоритм LSC по созданию суперпикселей

Листинг программы

```
import cv2
#Функция наложения маски контуров на изображение
```

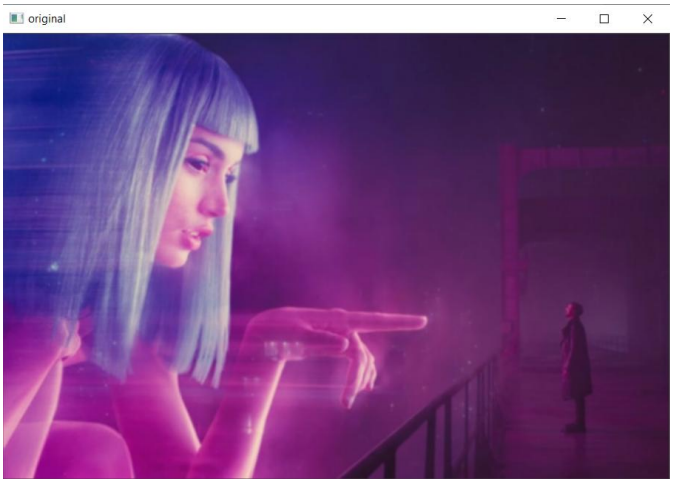
```
def workOnImage(superpixelStructure):
    contourMask = superpixelStructure.getLabelContourMask()
    invertedMask = cv2.bitwise_not(contourMask)
    return cv2.bitwise_and(image, image, mask=invertedMask)

#Читаем изображение и используем на нем размытие Гаусса для эффективного разбиения
image = cv2.imread('bladerunner.jpg', 1)
image = cv2.GaussianBlur(image, (3, 3), cv2.BORDER_CONSTANT)
cv2.imshow('original', image)

#Разбиваем изображение на суперпиксели алгоритмом SEEDS
SEEDS = cv2.ximgproc.createSuperpixelSEEDS(image.shape[1], image.shape[0],
image.shape[2], 2000, 20, 5, 1, True)
SEEDS.iterate(image, 20)
SEEDSImage = workOnImage(SEEDS)
cv2.imshow('SEEDSImage', SEEDSImage)

#Разбиваем изображение на суперпиксели алгоритмом SLIC
SLIC = cv2.ximgproc.createSuperpixelSLIC(image, cv2.ximgproc.SLIC, 10, 0.075)
SLIC.iterate(20)
SLICImage = workOnImage(SLIC)
cv2.imshow('SLICImage', SLICImage)

#Разбиваем изображение на суперпиксели алгоритмом SLICO
SLICO = cv2.ximgproc.createSuperpixelSLIC(image, cv2.ximgproc.SLICO, 10, 0.075)
SLICO.iterate(20)
SLICOImage = workOnImage(SLICO)
cv2.imshow('SLICOImage', SLICOImage)

#Разбиваем изображение на суперпиксели алгоритмом MSLIC
MSLIC = cv2.ximgproc.createSuperpixelSLIC(image, cv2.ximgproc.MSLIC, 10, 0.075)
MSLIC.iterate(20)
MSLICImage = workOnImage(MSLIC)
cv2.imshow('MSLICImage', MSLICImage)

#Ждем нажатия клавиши Escape, чтобы закрыть все окна
while (1):
    c = cv2.waitKey(33)
    if (c == 27):
        break
cv2.destroyAllWindows()
```

Программа реализации алгоритма LSC

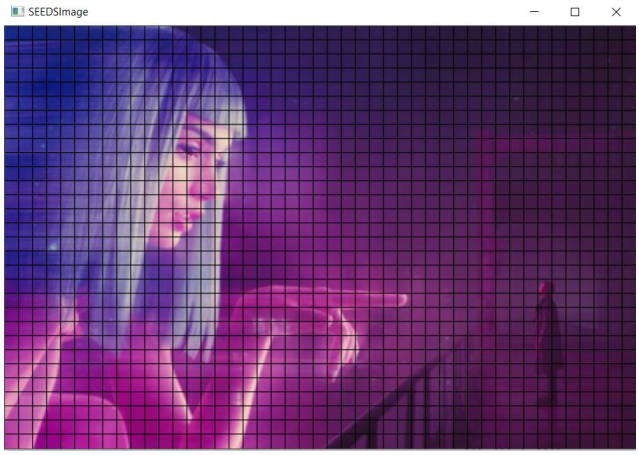
```
import cv2
img = cv2.imread("a.jpg")
//Функция по созданию алгоритма LSC
lsc = cv2.ximgproc.createSuperpixelLSC(img)
lsc.iterate(10)
maskLsc = lsc.getLabelContourMask()
labelLsc = lsc.getLabels()
numberLsc = lsc.getNumberOfSuperpixels()
maskInvLsc = cv2.bitwise_not(maskLsc)
imageLsc = cv2.bitwise_and(img, img, mask = maskInvLsc)
cv2.imshow("imgLsc", imageLsc)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Результаты работы

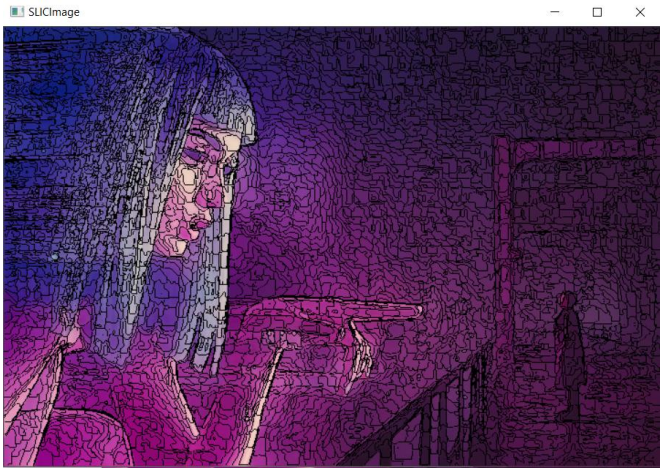
Оригинальное изображение



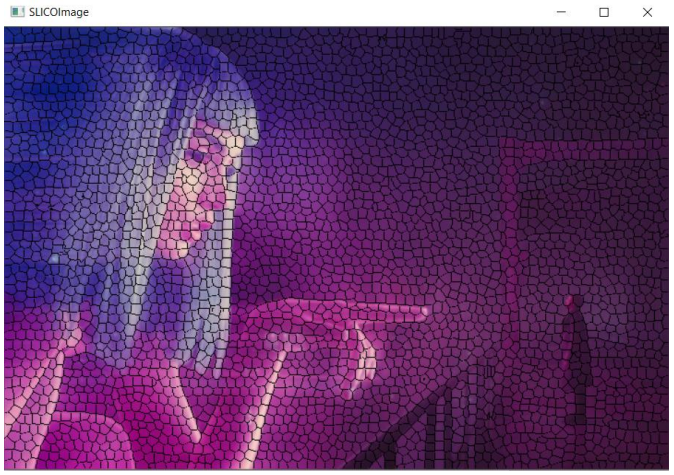
Изображение после алгоритма SEEDS



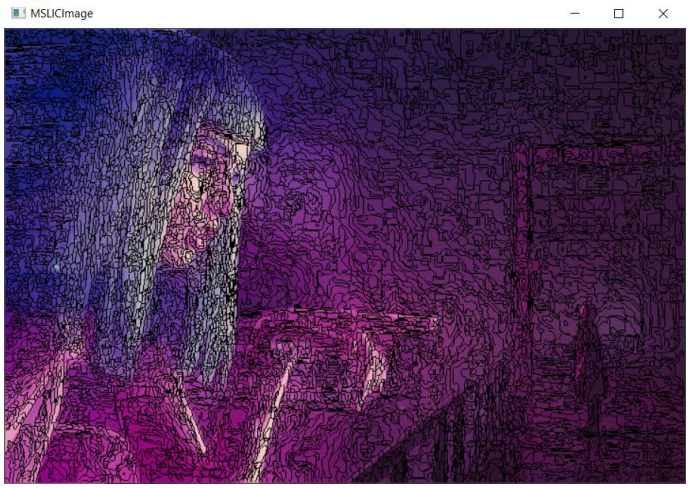
Изображение после алгоритма SLIC



Изображение после алгоритма SLICO



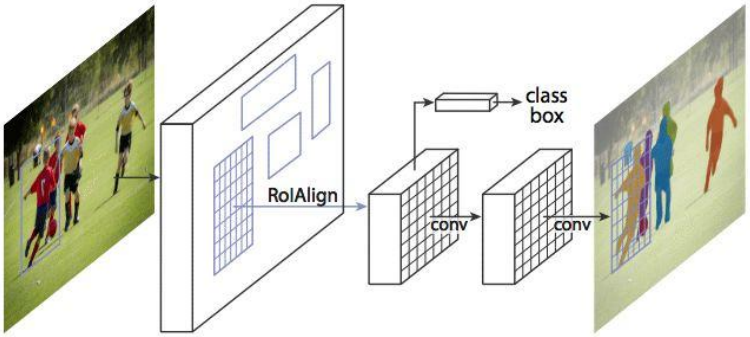
Изображение после алгоритма MSLIC

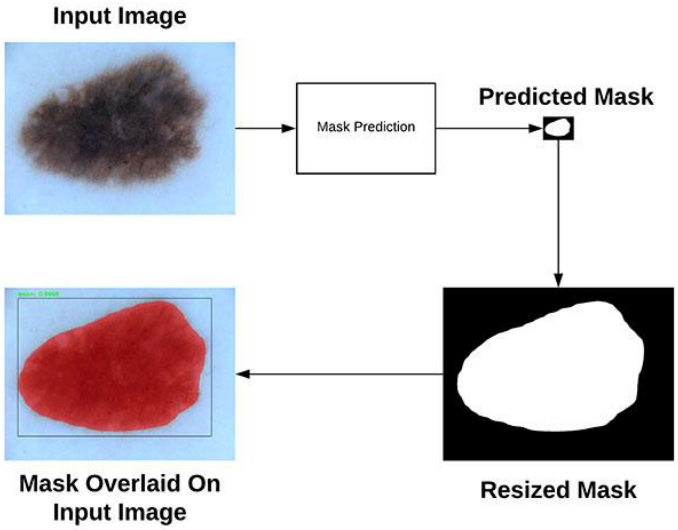


3.10. Применение нейронной сети Mask RCNN для сегментирования

Mask RCNN – глубокая нейронная сеть, предназначенная для решения проблемы сегментации экземпляров в машинном обучении или компьютерном зрении.

Эта сеть работает в два этапа: 1) генерирует предположения о регионах, где может находиться объект на основе входного изображения; 2) предсказывает класс объекта, уточняет ограничивающую рамку и генерирует маску на уровни пикселей объекта на предположениях первого этапа.





Алгоритм реализован при помощи нескольких функций: readNetFromTensorflow(), blobFromImage(), forward(), do_postprocess_work(), print_box(). Функция readNetFromTensorflow() считывает модель нейронной сети, которая была заготовлена заранее. Функция blobFromImage() создает 4-х мерный объект (первое измерение представляет количество обнаруженных блоков в кадре, второе измерение представляет количество классов в модели, а третье и четвертое измерения представляют форму маски). Функция forward() нужна, чтобы выполнить прямой проход для получения списка прогнозируемых ограничивающих рамок и масок объектов из выходных слоев. Функция do_postprocess_work() необходима для извлечения ограничивающей рамки и маски для каждого обнаруженного объекта, а функция print_box() нужна для рисовки предсказанной маски, ее покраски и показа на выходном изображении.

Формат функций OpenCV:

readNetFromTensorflow(model, config) – считывает модель сети, с помощью модель и конфигурации, которую взяли из открытого доступа.

blobFromImage(frame, swapRB, crop) – создает 4-х мерный объект из кадра, этот объект используется для того, чтобы вызвать функцию

forward([outputBlobs, outBlobNames]) – функция, проходящая по выходным слоям и для получения рамок и масок каждого объекта.

do_postprocess_work(boxes, masks) – извлекает маски и рамки, а

print_box(box_of_frame, id_of_class, trust, left, top, right, bottom, mask_of_class) – уже рисует маски и рамки на самом изображнии.

- model подготовленные модели
- config конфигурация
- frame кадр, который нужно обработать
- swapRB, crop две переменные логического типа: одна отвечает за смену красного и синего, вторая за обрезание изображния
- outputBlobs, outBlobNames два параметра Map, которые нужны для получения данных
- boxes, masks эти переменные, отвечающие за рамки и маски, получили при прямом проходе
- box_of_frame кадр
- id_of_class номер класса модели
- trust уровень доверия
- left, top, right, bottom положение маски
- mask_of_class маска класса

Программа выполняет следующие действия:

Подключает 6 библиотек: opencv, argpase, numpy, os.path, sys, random.

Загружает имена классов, загружают сеть, выбирает девайс для обработки (CPU или GPU), выбирает изображение(картинка, видео, видео записывающееся на web камеру) на основе входных параметров.

В цикле создает 4-х мерный объект из захваченного изображения, делает прямой проход.

Далее используются основные программы do_postprocess_work и print_box и с помощью функции imshow отображается получевшееся изображение.

Использованные функции

Функция	Описание
ArgumentParser(), add_argument()	Дают возможность использование параметров при запуске программы.
readNetFromTensorflow(weightsOfModel, textGraph)	Считывает модель и конфигурацию для нейронной сети.
blobFromImage(frame, swapRB=True, crop=False)	Возвращает 4-х мерный объект (binaryLargeObject) из нашего изображения.
setInput(binaryLargeObject)	Загружает в нейронную сеть наш 4-х мерный объект
do_postprocess_work(boxes, masks)	Выполняет для каждого кадра извлекает маску и рамку для каждого обнаруженного объекта
print_box(frame, id_of_class, score, left, top, right, bottom, mask_of_class)	Рисует на изображении рамку и маску для объектов
imwrite(outputFile, frame.astype(np.uint8)) write(frame.astype(np.uint8)) imshow(nameOfWindow, frame)	Записываем результат в статическое изображение или на видео или на изображение с веб-камеры, снимающей в реальном времени
waitKey(1)	Ждет нажатия клавиши 1 на клавиатуре. Как только клавиша будет нажата, работа программы прервется

Листинг программы

```
import cv2 as computer_vision
import argparse
import numpy as np
import os.path
import sys
import random

# Инициализируем параметры
limit_of_trust = 0.5  # Порог доверия
limit_of_mask = 0.3  # Порог маски

parser = argparse.ArgumentParser(description='Use this script to Mask-RCNN object
detection and segmentation')
parser.add_argument('--image', help='Image file')
parser.add_argument('--video', help='Video file.')
parser.add_argument("--device", default="cpu", help="The device that will be used
for work")
args = parser.parse_args()
```

```
# Рисуем предсказанную ограничивающую рамку, раскаршиваем ее и показываем
def print_box(box_of_frame, id_of_class, trust, left, top, right, bottom,
mask_of_class):
    # Рисуем ограничивающей маски
    computer_vision.rectangle(box_of_frame, (left, top), (right, bottom), (255,
178, 50), 3)

    # Печатаем метку класса
    label = '%.2f' % trust
    if allClasses:
        assert (id_of_class < len(allClasses))
        label = '%s:%s' % (allClasses[id_of_class], label)

    # Отображаем метки в верхней части ограничительной рамки
    size_of_label, line_of_base = computer_vision.getTextSize(label,
computer_vision.FONT_HERSHEY_SIMPLEX, 0.5, 1)
    top = max(top, size_of_label[1])
    computer_vision.rectangle(box_of_frame, (left, top - round(1.5 *
size_of_label[1])),
                              (left + round(1.5 * size_of_label[0]), top +
line_of_base), (255, 255, 255), computer_vision.FILLED)
    computer_vision.putText(box_of_frame, label, (left, top),
computer_vision.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 0), 1)

    # Изменяем размер маски, порог, цвета и применяем его к изображению
    mask_of_class = computer_vision.resize(mask_of_class, (right - left + 1,
bottom - top + 1))
    mask = (mask_of_class > limit_of_mask)
    roi = box_of_frame[top:bottom + 1, left:right + 1][mask]

    # Создаем разные цвета экземпляра
    index_of_color = random.randint(0, len(colors) - 1)
    color = colors[index_of_color]


box_of_frame[top:bottom + 1, left:right + 1][mask] = (
                [0.3 * color[0], 0.3 * color[1], 0.3 * color[2]] + 0.7 *
roi).astype(np.uint8)

    # Рисуем контура на изображении
    mask = mask.astype(np.uint8)
    contours, hierarchical = computer_vision.findContours(mask,
computer_vision.RETR_TREE, computer_vision.CHAIN_APPROX_SIMPLE)
    computer_vision.drawContours(box_of_frame[top:bottom + 1, left:right + 1],
contours, -1, color, 3, computer_vision.LINE_8, hierarchical, 100)


# Для каждого кадра извлекаем ограничивающую рамку и маску для каждого
обнаруженного объекта
def do_postprocess_work(boxes, masks):
    # Выходной размер масок равен NxCxHxW, где
    # N - количество обнаруженных боксов
    # C - количество классов (без учета фона)
    # HxW - форма сегментации

    num_of_detections = boxes.shape[2]

    frame_of_height = frame.shape[0]
```

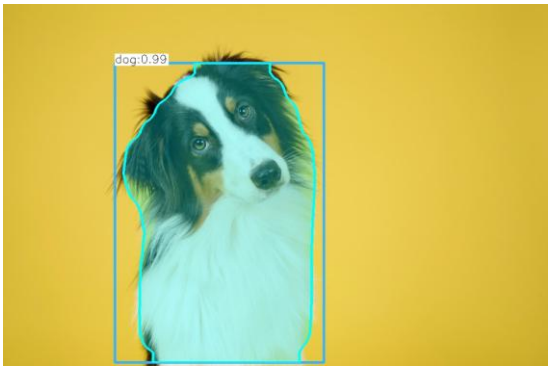
```
    frame_of_width = frame.shape[1]

    for elem in range(num_of_detections):
        box = boxes[0, 0, elem]
        mask = masks[elem]
        score = box[2]
        if score > limit_of_trust:
            id_of_class = int(box[1])

            # Извлекаем ограничительную рамку
            bottom = int(frame_of_height * box[6])
            right = int(frame_of_width * box[5])
            top = int(frame_of_height * box[4])
            left = int(frame_of_width * box[3])

            bottom = max(0, min(bottom, frame_of_height - 1))
            right = max(0, min(right, frame_of_width - 1))
            top = max(0, min(top, frame_of_height - 1))
            left = max(0, min(left, frame_of_width - 1))

            # Извлекаем маску для объекта
            mask_of_class = mask[id_of_class]

            # Рисуем ограничивающую рамку, раскрашиваем и показываем маску на
изображение
            print_box(frame, id_of_class, score, left, top, right, bottom,
mask_of_class)


# Загружаем имена классов
fileOfClasses = "mscoco_labels.names"
allClasses = None
with open(fileOfClasses, 'rt') as f:
    allClasses = f.read().rstrip('\n').split('\n')

# Загружаем файлы textGraph и weight для модели
textGraph = "./mask_rcnn_inception_v2_coco_2018_01_28.pbtxt"
weightsOfModel =
"./mask_rcnn_inception_v2_coco_2018_01_28/frozen_inference_graph.pb"

# Загружаем сеть
net = computer_vision.dnn.readNetFromTensorflow(weightsOfModel, textGraph)

if args.device == "cpu":
    net.setPreferableBackend(computer_vision.dnn.DNN_TARGET_CPU)
    print("Now working: CPU")
elif args.device == "gpu":
    net.setPreferableBackend(computer_vision.dnn.DNN_BACKEND_CUDA)
    net.setPreferableTarget(computer_vision.dnn.DNN_TARGET_CUDA)
    print("Now working: GPU")

# Загружаем классы
colorsFile = "colors.txt"
with open(colorsFile, 'rt') as f:
    colorsStr = f.read().rstrip('\n').split('\n')
colors = []  # [0,0,0]
for i in range(len(colorsStr)):
    rgb = colorsStr[i].split(' ')
    color = np.array([float(rgb[0]), float(rgb[1]), float(rgb[2])])
    colors.append(color)
```

```
nameOfWindow = 'Example of Mask-RCNN'
computer_vision.namedWindow(nameOfWindow, computer_vision.WINDOW_NORMAL)

outputFile = "mask_rcnn_out .avi"
if args.image:
    # Открываем файл изображения
    if not os.path.isfile(args.image):
        print("File with image ", args.image, " doesn't exist")
        sys.exit(1)
    capture = computer_vision.VideoCapture(args.image)
    outputFile = args.image[:-4] + '_mask_rcnn_out.jpg'
elif args.video:
    # Открываем видео файл
    if not os.path.isfile(args.video):
        print("File with video ", args.video, " doesn't exist")
        sys.exit(1)
    capture = computer_vision.VideoCapture(args.video)
    outputFile = args.video[:-4] + '_mask_rcnn_out.avi'
else:
    # Включаем веб-камеру, если нет никаких входных параметров
    capture = computer_vision.VideoCapture(0)

# Инициализируем запись видео, чтобы сохранить выходное видео
if not args.image:
    video_writer = computer_vision.VideoWriter(outputFile,
computer_vision.VideoWriter_fourcc('M', 'J', 'P', 'G'), 30,

(round(capture.get(computer_vision.CAP_PROP_FRAME_WIDTH)),
round(capture.get(computer_vision.CAP_PROP_FRAME_HEIGHT))))

while computer_vision.waitKey(1) < 0:

    # Получаем кадрр
    hasFrame, frame = capture.read()

    # Останавливаем программу, если пришел конец видео
    if not hasFrame:
        print("Mask-RCNN process is done!!!!!!!!")
        print("Name of output file is ", outputFile)
        computer_vision.waitKey(3000)
        break

    # Создаем 4D-большой двоичный объект из кадра.
    binaryLargeObject = computer_vision.dnn.blobFromImage(frame, swapRB=True,
crop=False)

    # Устанавливаем вход для сети
    net.setInput(binaryLargeObject)

    # Выполняем прямой проход, чтобы получить выходные данные из выходных слоев
    boxes, masks = net.forward(['detection_out_final', 'detection_masks'])

    # Извлекаем ограничивающую рамку и маску для каждого из обнаруженных объектов
    do_postprocess_work(boxes, masks)

    # Записываем кадр с полями обнаружения
    if args.image:
        computer_vision.imwrite(outputFile, frame.astype(np.uint8))
    else:
```

```
        video_writer.write(frame.astype(np.uint8))

    computer_vision.imshow(nameOfWindow, frame)
```

Результаты программы:

Исходное изображение Результат алгоритма Mask-RCNN



Глава 4. Алгоритмы высококоуровневой обработки изображений

4.1. Стабилизация видео

Стабилизация видео представляется важной задачей как при просмотре видео с камер, так и позиционировании объектов в потоке видеокадров. Можно выделить три этапа в методах стабилизации:

- На первом этапе выполняется оценка смещения между соседними кадрами. Оценка смещения осуществляется по смещению особых точек на видео. Точность оценки имеет решающее значение.
- На следующем этапе на основе рассчитанных смещений формируется новая последовательность кадров.
- Затем проводится повышение качества видео путем дополнительной обработки (устранение дефектов, сглаживание, граничная экстраполяция, др.).
- На последнем этапе возможно использование механизма устранения размытости (deblurring). Эффект размытости является основным источником ухудшения качества изображения. Он вызван движением объекта в кадре во время съемки. Поэтому использование алгоритмов устранения размытости необходимо в большинстве случаев.

Для выполнения алгоритма в OpenCV имеется набор функций, которые можно использовать на разных этапах. Причем при реализации алгоритма требуется существенное процессорное время на вычисления. В OpenCV имеются функции, которые используют возможности технологии CUDA.

Алгоритм стабилизации видео

Листинг программы

```
# import required libraries
from vidgear.gears.stabilizer import Stabilizer
import cv2

# Open suitable video stream, such as webcam on first index(i.e. 0)
stream = cv2.VideoCapture('video_3.mp4')
# initiate stabilizer object with default parameters
stab = Stabilizer()
# loop over
while True:

    # read frames from stream
    (grabbed, frame) = stream.read()

    # check for frame if not grabbed
    if not grabbed:
        break

    # send current frame to stabilizer for processing
    stabilized_frame = stab.stabilize(frame)

    # wait for stabilizer which still be initializing
    if stabilized_frame is None:
        continue

        # {do something with the frame here}

    # Show output window
    cv2.imshow("Frame", frame)
    cv2.imshow("Stabilized frame", stabilized_frame)


    # check for 'q' key if pressed
    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        break
```

```
# close output window
cv2.destroyAllWindows()

# clear stabilizer resources
stab.clean()

# safely close video stream
stream.release()
```

4.2. Стабилизация видео с помощью FFmpeg

Стабилизация видео позволяет компенсировать движения и дрожание камеры на видеозаписи. Помимо технических приспособлений существуют также и программные средства для стабилизации видео в последующей обработке.

Фреймворк FFmpeg

FFmpeg — набор свободных библиотек с открытым исходным кодом, которые позволяют записывать, конвертировать, обрабатывать, и передавать цифровые аудио- и видеозаписи в различных форматах. FFmpeg изначально разработан на основе Linux, однако он предоставляет исходный код, который может быть скомпилирован под практически любую операционную систему. На официальном сайте FFmpeg можно найти ссылки на готовые сборки инструментов FFmpeg для Linux, Windows, и macOS вместе с инструкциями по их установке.

Для интеграции функций FFmpeg в язык программирования Python существует множество оберток, одна из самых популярных, позволяющих поддерживать сложные фильтры и преобразования это ffmpeg-python, распространяемая по свободной лицензии Apache Licencse 2.0.

Для установки этой библиотеки можно воспользоваться pip (система управления пакетами в Питоне):

pip install ffmpeg-python

Фильтры стабилизации

Стабилизация видео средствами FFmpeg проходит в три этапа. Фильтр vidstabdetect анализирует тряску видео путем оценки смещения между соседними кадрами, вычисления особых точек и формирования матрицы трансформаций. Он генерирует выходной файл, описывающий смещения для использования в следующем этапе. У этого фильтра есть ряд параметров, в программе используются следующие:

1. Shakiness – условный показатель степени тряски видео, целое число от 1 до 10
2. Accuracy – показатель точности определения смещения, целое число от 1 до 15
3. Result – указывает путь к выходному файлу с описаниями трансформаций

Фильтр vidstabtrasform считывает описания трансфорамаций, полученные в предыдущем этапе и синтезирует стабильную картинку. В программе используются следующие параметры:

Input – указывает на файл с трансформациями

Smoothing – задает количество кадров, по которому происходит сглаживание. Принимает число и вычисляет количество кадров как (значение * 2) + 1. Большее число кадров дает более гладкое видео, но ограничивает диапазон перемещения и вращения камеры.

Фильтр unsharp создан для увеличения или уменьшения резкости, здесь используется для устранения размытости объектов в полученном кадре. Он принимает параметры luma (яркостный компонент YUV-палитры) и chroma (цветоразностные компоненты) матриц, а также силу luma и chroma сглаживания. Значения этих параметров задаются для цифровой субдискретизации изображения и установки степени сглаживания, большие значения дают позволяют фильтру производить

более точную обработку, и в результате приводят к более качественному результату, однако они сильно влияют на производительность

Значения и дополнительные параметры фильтров настраиваются индивидуально исходя из специфики обрабатываемых видео. Тщательная и тонкая настройка позволит получить впечатляющие результаты обработки.

Программа также производит склеивание исходного и финального видео друг рядом с другом для наглядного сравнения, для этого используется фильтр hstack фреймворка FFmpeg.

Использованные функции обертки ffmpeg-python (Используемая обертка формирует команды для FFmpeg с помощью отдельных функций)

input()	Задает входной файл команды
filter()	Применяет указанный фильтр, в функцию можно передать опциональные именные аргументы и они преобразуются в параметры фильтра
output()	Задает выходной файл
overwrite_output()	Позволяет перезаписывать существующие файлы при повторном выполнении
run()	Собирает, формирует и выполняет команду для FFmpeg

Листинг программы

```
import ffmpeg
import sys


def ffmpeg_stab(in_filename, out_filename, merged_filename):
    try:
        # выполняем vidstabdetect
        (
            ffmpeg
            .input(in_filename)
            .filter('vidstabdetect', shakiness=10, accuracy=15, result="t.trf")
            .output(out_filename)
            .overwrite_output()
            .run()
        )
        # выполняем vidstabtransform и unsharp
        (
            ffmpeg
            .input(in_filename)
            .filter('vidstabtransform', input="t.trf",smoothing=20)
```

```
            .filter('unsharp', luma_msize_x=9, luma_msize_y=9, luma_amount=1.1,
chroma_msize_x=7, chroma_msize_y=7, chroma_amount=0.7)
            .output(out_filename)
            .overwrite_output()
            .run()
        )
        # берем исходное и получившееся видео и склеиваем их вместе
        orig = ffmpeg.input(in_filename)
        stab = ffmpeg.input(out_filename)
        (
            ffmpeg
            .filter([orig, stab], 'hstack')
            .output(merged_filename)
            .overwrite_output()
            .run()
        )

    except ffmpeg.Error as e:
        print(e.stderr.decode(), file=sys.stderr)
        sys.exit(1)


if __name__ == '__main__':
    ffmpeg_stab("source.mp4", "ffmpegstab.mp4", "ffmpegstabmerged.mp4")
```

Результаты

На выходе получается стабилизированное видео и сравнительное. В сравнительном видео слева поставлено оригинальное, а справа обработанное, можно заметить, что картинка на обработанном видео немного приближена вследствие обрезания краев смещенных кадров.

4.3. Отслеживание объекта. Трекер

Отслеживание позволяет просто следить за передвижением объекта, не рассматривая его сущность. Отслеживание работает быстрее, чем обнаружение. Когда вы отслеживаете объект, обнаруженный в предыдущем кадре, вы многое знаете о внешнем виде объекта. Вы также

знаете местоположение в предыдущем кадре, направление и скорость его движения. Поэтому в следующем кадре вы можете использовать всю эту информацию для прогнозирования местоположения объекта. Хороший алгоритм отслеживания может работать, когда обнаружение невозможно. Если важная часть объекта ушла из поля зрения, трекинг алгоритм может все равно знать о местоположении объекта, основываясь на предшествующей информации.

OpenCV включает в себя 5 трекеров: BOOSTING, MIL, KCF, TLD, MEDIANFLOW.

BOOSTING Tracker. Трекер, основанный на каскаде Хаара. Первое выделение объекта – первый положительный пример. С дальнейшей работой алгоритма записывается все больше положительных примеров для отслеживаемого объекта.

MIL Tracker. Большая разница заключается в том, что вместо того, чтобы рассматривать только текущее местоположение объекта как положительный пример, он развертывается в небольшом окружении вокруг текущего местоположения, чтобы генерировать несколько потенциальных положительных примеров.

KCF Tracker. Объединяет в себе идеи двух предыдущий трекеров. Этот трекер использует тот факт, что множественные положительные образцы, используемые в трекер MIL, имеют большие перекрывающиеся области. Эти перекрывающиеся области позволяют получить некоторые математические свойства, которые используют этот трекер, чтобы сделать отслеживание быстрее и точнее в одно и то же время.

TLD Tracker. Расшифровка аббревиатуры – отслеживание, обучение, обнаружение. Трекер следует за объектом от кадра до кадра. Детектор

локализует все видимые явления, которые наблюдались до сих пор, и при необходимости корректирует трекер.

MEDIANFLOW Tracker. Отслеживание за объектом происходит путём измерения расхождения между прямой и обратной траекторией (направлением) движения.

Листинг программы

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import matplotlib
import copy

matplotlib.use('Qt5Agg')
import numpy
from scipy.ndimage import label

exit = False
while not exit:
    cap1 = cv2.VideoCapture('tracking.mp4')
    while True:
        ret1, frame1 = cap1.read()
        if frame1 is None:
            break

        # render
        cv2.imshow('ORIGINAL', frame1)
        cv2.moveWindow("ORIGINAL", 0, 0)

        # handle keys
        key = cv2.waitKey(0) & 0xFF
        if key == ord('d'):
            ret1, frame1 = cap1.read()
        elif key == ord('q'):
            exit = True
            break
        elif key == ord('s'):
            break
    if exit:
        break

    # Track Init
    ret1, frame1 = cap1.read()

    region = cv2.selectROI("Select region", frame1, False)
    if (not region):
        quit()

    # coord = region
    # p1 = (int(coord[0]), int(coord[1]))
    # p2 = (int(coord[0] + coord[2]), int(coord[1] + coord[3]))
    # frame_region = frame1
    # cv2.rectangle(frame_region, p1, p2, (0, 255, 0), 2, 1)
    # cv2.imshow('FRAME', frame_region)
```

```
# cv2.moveWindow("FRAME", 0, 500)
cv2.destroyWindow("Select region")
cv2.destroyWindow("ORIGINAL")

# init kcf tracker
trackers = {
    "KCF": cv2.TrackerKCF_create(),
    "BOOSTING": cv2.TrackerBoosting_create(),
    "MIL": cv2.TrackerMIL_create(),
    "TLD": cv2.TrackerTLD_create(),
    "MEDIAN_FLOW": cv2.TrackerMedianFlow_create(),
}
status = [trackers[k].init(frame1, region) for k in trackers]
if not all(status):
    print('ERROR')
    quit()

coord = [region for i in range(0, 5)]
draw_coord = [region for i in range(0, 5)]
isWorking = [True for i in range(0, 5)]
####

exit = False
while not exit:
    cap1 = cv2.VideoCapture('tracking.mp4')
    while True:
        ret1, frame1 = cap1.read()
        if frame1 is None:
            break

        cv2.imshow('ORIGINAL', frame1)
        cv2.moveWindow("ORIGINAL", 0, 0)

        # track
        if isWorking:
            for k in trackers:
                frame = copy.copy(frame1)
                status, coord = trackers[k].update(frame)
                if status and draw_coord:
                    p1 = (int(coord[0]), int(coord[1]))
                    p2 = (int(coord[0] + coord[2]), int(coord[1] + coord[3]))
                    cv2.rectangle(frame, p1, p2, (0, 255, 0), 2, 1)

                if k == "KCF":
                    cv2.imshow('KCF TRACKER', frame)
                    cv2.moveWindow("KCF TRACKER", 650, 0)
                elif k == "BOOSTING":
                    cv2.imshow('BOOSTING TRACKER', frame)
                    cv2.moveWindow("BOOSTING TRACKER", 1500, 0)
                elif k == "MIL":
                    cv2.imshow('MIL TRACKER', frame)
                    cv2.moveWindow("MIL TRACKER", 650, 480)
                elif k == "TLD":
                    cv2.imshow('TLD TRACKER', frame)
                    cv2.moveWindow("TLD TRACKER", 1500, 480)
                elif k == "MEDIAN_FLOW":
                    cv2.imshow('MEDIAN_FLOW TRACKER', frame)
                    cv2.moveWindow("MEDIAN_FLOW TRACKER", 0, 500)

        # handle keys
```

```
            key = cv2.waitKey(0) & 0xFF
            if key == ord('d'):
                ret1, frame1 = cap1.read()
            elif key == ord('q'):
                exit = True
                break
        if exit:
            break
cap1.release()
cv2.destroyAllWindows()
```

4.4. Отслеживание объекта, применение метода среднего сдвига

Средний сдвиг - это метод непараметрического пространственного анализа, так называемый алгоритм поиска режима. Это процедура для определения местоположения максимумов функции плотности с учетом дискретных данных, взятых из этой функции. В некотором смысле он использует непараметрическую оценку градиента плотности.

Основная функция:

cv.meanShift(prob_image, window, criteria) - находит объект на изображении обратной проекции в окне интереса.

Важное понятие в OpenCV - регион интересов ROI (Region Of Interest). Почти все функции должны поддерживать работу с ROI, т.е. работу с выделенной областью изображения, что полезно для ускорения работы алгоритмов.

Вот какие функции для работы с ROI:

cvSetImageROI(IplImage* image, CvRect rect) — установка интересующей области рисунка;

cvResetImageROI(IplImage* image) — сбрасывает область интересов;

cvGetImageROI(const IplImage* image) — возвращает область интересов изображения.

Для изучения метода разработана программа, которая выполняет следующие действия:

1. Загружает видео из заданного файла.

2. Устанавливает начальные расположения окна
3. Настраивает ROI для отслеживания
4. Устанавливает критерии завершения, либо 10 итераций, либо переместитесь как минимум на 1 пункт
5. Далее в цикле:
6. Конвертирует изображение BGR в HSV, чтобы мы могли использовать его для извлечения цветного объекта. В HSV легче представить цвет, чем в RGB.
7. Применяет meanShift чтобы получить новое расположение.
8. Вывод изображение на экран.

Использованные функции

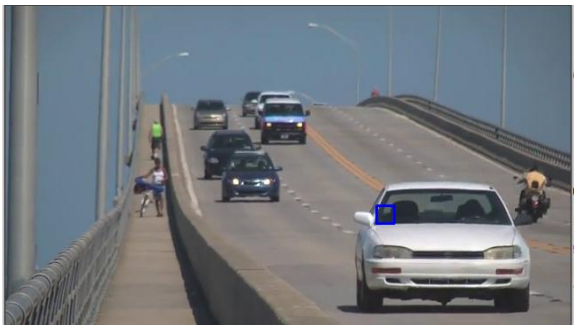
cap.read()	Если кадр прочитан правильно, вернет true, иначе false
cv2.waitKey()	Чем меньше аргумент, тем быстрее идет видео, и наоборот
cv2.inRange()	Используется для установки порога HSV для получения определенного цвета
calcHist()	Вычисляет гистограмму одного или нескольких массивов. Элементы кортежа, используемые для увеличения ячейки гистограммы, берутся из соответствующих входных массивов в том же месте.
calcBackProject()	Вычисляет задний проект гистограммы. То есть, как и calcHist (), в каждом месте (x, y) функция собирает значения из выбранных каналов во входных изображениях и находит соответствующий интервал гистограммы.
cv.meanShift(prob_image, window, criteria)	Находит объект на изображении обратной проекции. probImage - обратная проекция гистограммы объекта. window - начальное окно поиска. criteria - критерии остановки для алгоритма итеративного поиска.

Листинг программы

```
import numpy as np
import cv2

cap = cv2.VideoCapture('test.mp4')

# take first frame of the video
ret,frame = cap.read()

# setup initial location of window
# r,h,c,w - region of image
#           simply hardcoded the values
r,h,c,w = 200,20,300,20
track_window = (c,r,w,h)

# set up the ROI for tracking
roi = frame[r:r+h, c:c+w]
hsv_roi =  cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)

# Setup the termination criteria, either 10 iteration or move by at least 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )

while(1):
    ret ,frame = cap.read()

    if ret == True:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

        # apply meanshift to get the new location
        ret, track_window = cv2.meanShift(dst, track_window, term_crit)

        # Draw it on image
        x,y,w,h = track_window
        img2 = cv2.rectangle(frame, (x,y), (x+w,y+h), 255,2)
        cv2.imshow('img2',img2)

        k = cv2.waitKey(60) & 0xff
        if k == 27:
            break
        else:
            cv2.imwrite(chr(k)+".jpg",img2)

    else:
        break

cv2.destroyAllWindows()
cap.release()
```

Результаты работы программы






4.5. Определение жеста

Алгоритм определения жеста включает следующие этапы:

1) Конвертируем изображение в RGB, только с таким форматом работает библиотека HandTrackingModule;

2) Проверяем сколько рук обнаружено и запоминаем координаты;

3) Каждая модель руки состоит из 21 точки;

4) Нам требуется найти точки 8 и 4, между которыми определяется расстояние (жест увеличения);

5) Рисуем линию, соединяющую точки 8 и 4;

6) Рисуем заполненные круги на точки 8 и 4 и на точку в середине отрезка, соединяющего 8 и 4;

7) Находим минимальную и максимальную длину линии, между точками 8 и 4, добавляем цвет на линию, при длине, равной минимальной;

8) Рисуем прямоугольник, который в зависимости от длинны линии между точками 4 и 8, раскрашивается в процентном соотношении и добавляем надпись процентов;

9) Добавляем счетчик количества кадров в секунду, для этого берем текущее время и отнимаем время предыдущего кадра, и делим 1 на результат, выводим, как текст, на картинку.

Список использованных функций OpenCV

Наименование функции	Краткое описание
absdiff	Вычисляет абсолютную разницу для каждого элемента между двумя массивами или скалярами.
cvtColor	Изменение цветовой палитры изображения
GaussianBlur	Применяет двусторонний фильтр к изображению
threshold	Функция используется для определения порога.
dilate	Набор операций, обрабатывающих изображений на основе фигур
findContours	Инструмент для анализа формы и распознования объекта
imshow	Создает окно и демонстрирует изображение

Программа

Gesture.py

```
import cv2
import time
import numpy as np
import HandTrackingModule as htm
import math
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

################################
wCam, hCam = 640, 480
################################
cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
pTime = 0
```

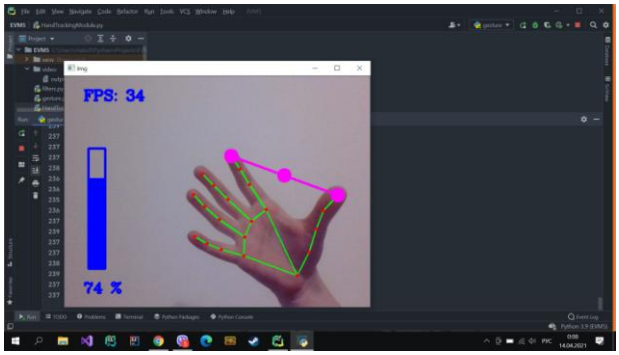
```
detector = htm.handDetector(detectionCon=0.7)

devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(
    IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
# volume.GetMute()
# volume.GetMasterVolumeLevel()
volRange = volume.GetVolumeRange()
minVol = volRange[0]
maxVol = volRange[1]
vol = 0
volBar = 400
volPer = 0
while True:
    success, img = cap.read()
    img = detector.findHands(img)
    lmList = detector.findPosition(img, draw=False)
    if len(lmList) != 0:
        # print(lmList[4], lmList[8])

        x1, y1 = lmList[4][1], lmList[4][2]
        x2, y2 = lmList[8][1], lmList[8][2]
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

        cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
        cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
        cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
        cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)

        length = math.hypot(x2 - x1, y2 - y1)
        # print(length)

        # Hand range 50 - 300
        # Volume Range -65 - 0

        vol = np.interp(length, [50, 300], [minVol, maxVol])
        volBar = np.interp(length, [50, 300], [400, 150])
        volPer = np.interp(length, [50, 300], [0, 100])
        print(int(length), vol)
        volume.SetMasterVolumeLevel(vol, None)

        if length < 50:
            cv2.circle(img, (cx, cy), 15, (0, 255, 0), cv2.FILLED)

    cv2.rectangle(img, (50, 150), (85, 400), (255, 0, 0), 3)
    cv2.rectangle(img, (50, int(volBar)), (85, 400), (255, 0, 0), cv2.FILLED)
    cv2.putText(img, f'{int(volPer)} %', (40, 450), cv2.FONT_HERSHEY_COMPLEX,
                1, (255, 0, 0), 3)

    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    cv2.putText(img, f'FPS: {int(fps)}', (40, 50), cv2.FONT_HERSHEY_COMPLEX,
                1, (255, 0, 0), 3)

    cv2.imshow("Img", img)
    cv2.waitKey(1)
```

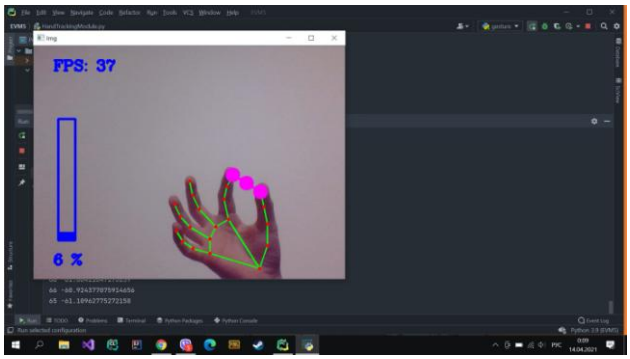
HandTrackingModule.py

```
import cv2
import mediapipe as mp
import time

class handDetector():
    def __init__(self, mode=False, maxHands=2, detectionCon=0.5, trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.trackCon = trackCon

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands,
                                        self.detectionCon, self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils

    def findHands(self, img, draw=True):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
        # print(results.multi_hand_landmarks)

        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms,
                                               self.mpHands.HAND_CONNECTIONS)
        return img

    def findPosition(self, img, handNo=0, draw=True):

        lmList = []
        if self.results.multi_hand_landmarks:
            myHand = self.results.multi_hand_landmarks[handNo]
            for id, lm in enumerate(myHand.landmark):
                # print(id, lm)
                h, w, c = img.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                # print(id, cx, cy)
                lmList.append([id, cx, cy])
                if draw:
                    cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)

        return lmList

def main():
    pTime = 0
    cTime = 0
    cap = cv2.VideoCapture(1)
    detector = handDetector()
    while True:
        success, img = cap.read()
        img = detector.findHands(img)
        lmList = detector.findPosition(img)
        if len(lmList) != 0:
            print(lmList[4])

        cTime = time.time()
        fps = 1 / (cTime - pTime)
        pTime = cTime
```

```
            cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3,
                        (255, 0, 255), 3)

            cv2.imshow("Image", img)
            cv2.waitKey(1)


    if __name__ == "__main__":
        main()
```

Скриншоты





4.6. Анализ частичного и полного оптического потока между кадрами изображения

Оптический поток — это картина видимого движения объектов изображения между двумя последовательными кадрами, вызванного движением объекта или камеры. Это двумерное векторное поле, где каждый вектор представляет собой вектор смещения, показывающий перемещение точек от первого кадра ко второму.

Оптический поток имеет множество применений в таких областях, как: структура из движущегося изображения; сжатие видео; стабилизация видео.

Идея оптического потока может быть основана на нескольких положениях: интенсивность пикселей не меняется между двумя соседними кадрами и соседние пиксели имеют одинаковое движение.

В нашем примере мы рассмотрим функцию calcOpticalFlowPyrLK, которая основана на методе Лукаса-Канаде.

Метод Лукаса-Канаде

Ранее мы высказали предположение, что все соседние пиксели будут иметь одинаковое движение. Метод Лукаса-Канаде использует область 3x3 вокруг точки. Итак, все 9 точек имеют одинаковое движение. Мы можем найти (f_x, f_y, f_t) для этих 9 точек. Итак, теперь наша проблема сводится к решению 9 уравнений с двумя переопределенными неизвестными переменными. Лучшее решение получается методом наименьших квадратов. Ниже приведено окончательное решение, которое представляет собой два неизвестных уравнения с двумя уравнениями и решает их, чтобы получить решение.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}{}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}{}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix}$$

Итого, обычный Лукас-Канаде хорошо определяет маленькие сдвиги, такие, в рамках которых картинка похожа на свое линейное приближение. Чтобы с этим побороться, воспользуемся стандартным приемом CV – multi-scaling'ом: построим «пирамиду» изображений разного масштаба (почти всегда берется масштабирование в 2 раза по каждой оси, так проще считать) и пройдем по ним оптическим потоком от меньшего изображения к большему, тогда детектированный маленький сдвиг на маленьком изображении будет соответствовать большому сдвигу на большом изображении. На самом маленьком изображении мы обнаруживаем сдвиг не более 1–2 пикселей, а переходя от меньшего масштаба к большему, мы пользуемся результатом с предыдущего шага и уточняем значения сдвига. Собственно, в OpenCV его и реализует функция calcOptFlowPyrLK.

Описание программы:

Для начала мы используем функцию goodFeaturesToTrack, которая с помощью алгоритма Ши-Томази находит 100 самых явных угловых точек, за перемещением оптического потока в которых мы и будем наблюдать.

Затем при получение очередного кадра мы будем сравнивать его с предыдущим и находить оптические потоки с помощью функции calcOptFlowPyrLK, которые и отображает с помощью треков.

Используемые функции

VideoCapture – создать объект **VideoCapture**

cap.read() – получить следующий кадр, если он не был получен возвращает false, иначе true

cv2.cvtColor используется для преобразования изображения из одного цветового пространства в другое

goodFeaturesToTrack – Методом Ши-Томази находит самые явные углы изображения.

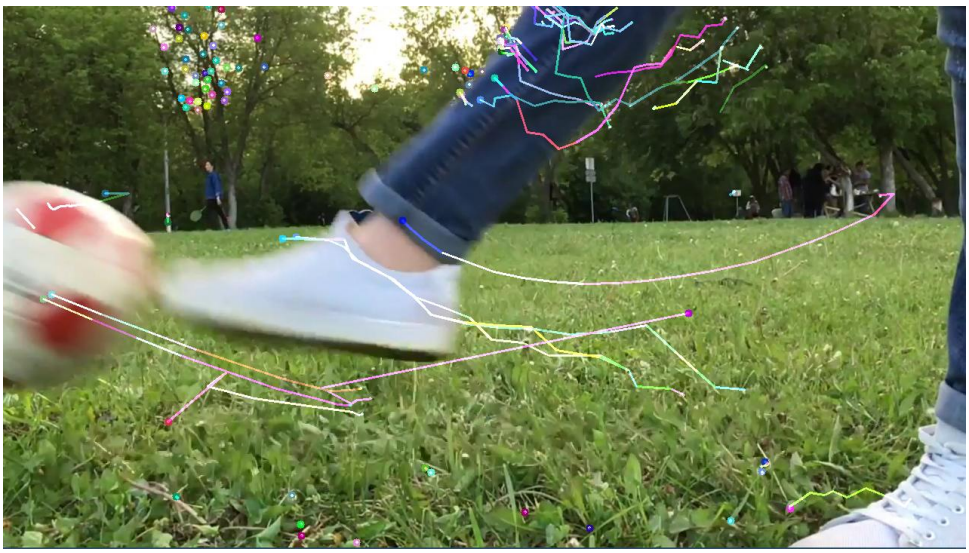
calcOpticalFlowPyrLK – методом Лукас-Канаде находим оптические потоки

Программа

```
import numpy as np
import cv2

cap = cv2.VideoCapture('ball.mp4')

# параметры для алгоритма Ши-Томази
shi_tomasi_params = dict(maxCorners = 100,
                         qualityLevel = 0.3,
                         minDistance = 7,
                         blockSize = 7)

# параметры для оптического потока
opticalflow_params = dict(winSize  = (15, 15),
                          maxLevel = 2,
                          criteria = (cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# сгенерировать случайные цвета
color = np.random.randint(0,255,(100,3))

# получить первый кадр и найти на нем углы алгоритмом Ши-Томази
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **shi_tomasi_params)

# создать маску для рисования
mask = np.zeros_like(old_frame)

while(1):
    ret,frame = cap.read()
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # вызвать функцию calcOpticalFlowPyrLK, чтобы найти оптический поток
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None,
**opticalflow_params)

    # выбрать новые точки
    if p1 is not None:
```

```
            good_new = p1[st==1]
            good_old = p0[st==1]

        # нарисовать треки
        for i,(new,old) in enumerate(zip(good_new,good_old)):
            a,b = new.ravel()
            c,d = old.ravel()
            mask = cv2.line(mask, (a,b),(c,d), color[i].tolist(), 2)
            frame = cv2.circle(frame,(a,b),5,color[i].tolist(),-1)
        img = cv2.add(frame,mask)

        cv2.imshow('image.jpg',img)
        k = cv2.waitKey(30) & 0xff
        if k == 27:
            break

        # обновить кадр и точки, за которыми следим
        old_gray = frame_gray.copy()
        p0 = good_new.reshape(-1,1,2)

cv2.destroyAllWindows()
cap.release()
```

Результат работы программы



4.7 Обобщенное преобразование Хафа и его применение для поиска объектов

Обобщённое преобразование Хафа (ОПХ) было предложено D.H. Ballard в 1981 году и представляет собой модификацию преобразования Хафа, использующую принципы сравнения шаблонов. Эта модификация позволяет использовать преобразование Хафа для обнаружения любого объекта, описываемого его моделью.

Задача нахождения объекта решается нахождением позиции модели на изображении. ОПХ преобразует задачу нахождения позиции модели в задачу нахождения параметров преобразования, которые отобразят модель на изображение. Зная значения параметров преобразования, можно определить положение модели на изображении.

В оригинальной реализации метода ОПХ использовались границы изображения для определения соотношения между ориентацией крайней точки и ориентира формы. В случае бинарного изображения, когда пиксели могут принимать только черный или белый цвет, каждый черный пиксель изображения может быть черным пикселем желаемого шаблона, таким образом создавая геометрическое место точек ориентира в пространстве Хафа. Каждый пиксель изображения голосует за соответствующие ему ориентиры изображения. Максимальное количество голосов в пространстве Хафа обозначает возможный ориентир шаблона в изображении. Это максимальное число можно найти с помощью сканирования пространства Хафа или решения математических уравнений, каждое из которых соответствует единичному черному пикселю.

Основная функция библиотеки OpenCV:

cv2.createGeneralizedHoughGuil() — возвращает объект GeneralizedHoughGuil, который впоследствии настраивается своими методами и использует метод detect(image) (где image — изображение шаблона в черно-белой палитре) для получения списка возможных позиций и списка с количеством голосов за каждую из этих позиций.

Некоторые методы для настройки параметров распознавания:

setTemplate(template) — загрузка шаблона для обнаружения

setAngleEpsilon(angleEpsilon) — максимальная погрешность между градусными мерами углов, которые будут считаться равными

setAngleStep(angleStep), setScaleStep(scaleStep) — шаг изменения угла и коэффициента подобия в градусах

setAngleThresh(angleThresh), setPosThresh(posThresh), setScaleThresh(scaleThresh) — количество голосов для «принятия» угла, позиции, коэффициента подобия

setMaxScale(maxScale), setMinScale(minScale) — максимальный и минимальный коэффициент подобия между шаблоном и найденным объектом

setMaxAngle(maxAngle), setMinAngle(minAngle) — максимальный и минимальный угол поворота фигуры при проверке

setMinDist(minDist) — минимальное расстояние между центрами найденных объектов

Алгоритм программы включает следующие этапы:

1. Подключение модулей OpenCV и numpy
2. Чтение исходного изображения и перевод его в ч/б палитру
3. Чтение шаблона и нахождение его границ через Canny
4. Создание объекта GeneralizedHoughGuil, его настройка и нахождение возможных положений шаблона
5. Рисование рамок найденных положений и вывод на экран
6. Сохранение результата в файл

Список использованных функций

Название функции	Описание
imread	Загрузка изображения из файла
cvtColor	Конвертация изображения из одной цветовой палитры в другую
Canny	Детектор ребер
createGeneralizedHoughGuil	Создание объекта для распознавания шаблона на изображении
boxPoints	Нахождение координат вершин повернутого

	прямоугольника
drawContours	Отображение контуров или контуров с заливкой
imwrite	Запись изображения в файл
imshow	Вывод изображения на экран
waitKey	Ожидание нажатия клавиши
destroyAllWindows	Закрытие всех созданных OpenCV окон

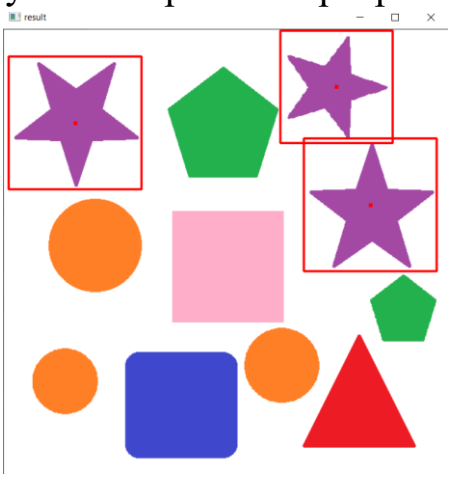
Листинг программы

```
import cv2
import numpy as np

# загрузка исходного изображения
img = cv2.imread("shapes.png")
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# загрузка шаблона и нахождение контуров на нем
template = cv2.imread("template.png")
height, width = template.shape[:2]
edges = cv2.Canny(template, 200, 250)

# создание объекта для обнаружения шаблона
ght = cv2.createGeneralizedHoughGuil()
ght.setTemplate(edges)
votes = 2000
ght.setAngleEpsilon(1)
ght.setAngleStep(1)
ght.setAngleThresh(votes * 150)
ght.setMaxAngle(360)
ght.setMaxScale(1.5)
ght.setMinAngle(0)
ght.setMinDist(100)
ght.setMinScale(0.5)
ght.setPosThresh(votes)
ght.setScaleStep(0.05)
ght.setScaleThresh(votes * 10)

# обнаружение положений найденных паттернов
positions, votes = (i[0] for i in ght.detect(img_gray))

# вывод на изображение рамок вокруг найденных паттернов
for position in positions:
    center_col = int(position[0])
    center_row = int(position[1])
    scale = position[2]
    angle = int(position[3])
    found_height = int(height * scale)
    found_width = int(width * scale)
    rectangle = ((center_col, center_row),
                 (found_width, found_height),
```

```
                angle)
    box = cv2.boxPoints(rectangle)
    box = np.int0(box)
    cv2.drawContours(img, [box], 0, (0, 0, 255), 2)
    # ... и вывод на изображение их центров
    for i in range(-2, 3):
        for j in range(-2, 3):
            img[center_row + i, center_col + j] = 0, 0, 255

# сохранение и вывод результата
cv2.imwrite("results.png", img)
cv2.imshow("template", edges)
cv2.imshow("result", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Скриншоты результатов работы программ

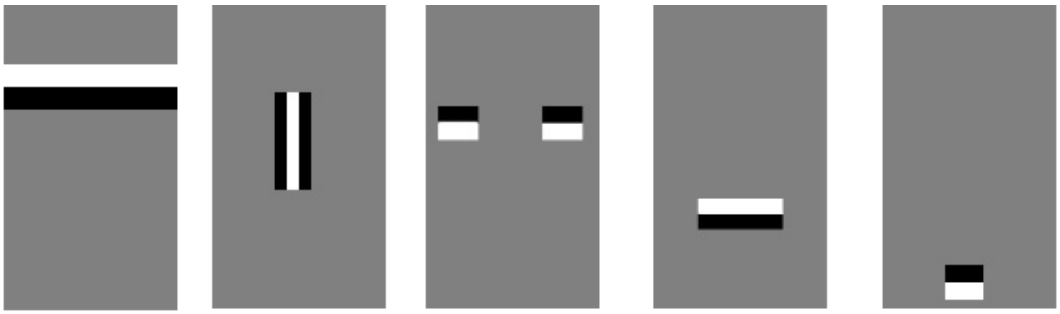
1.Шаблон



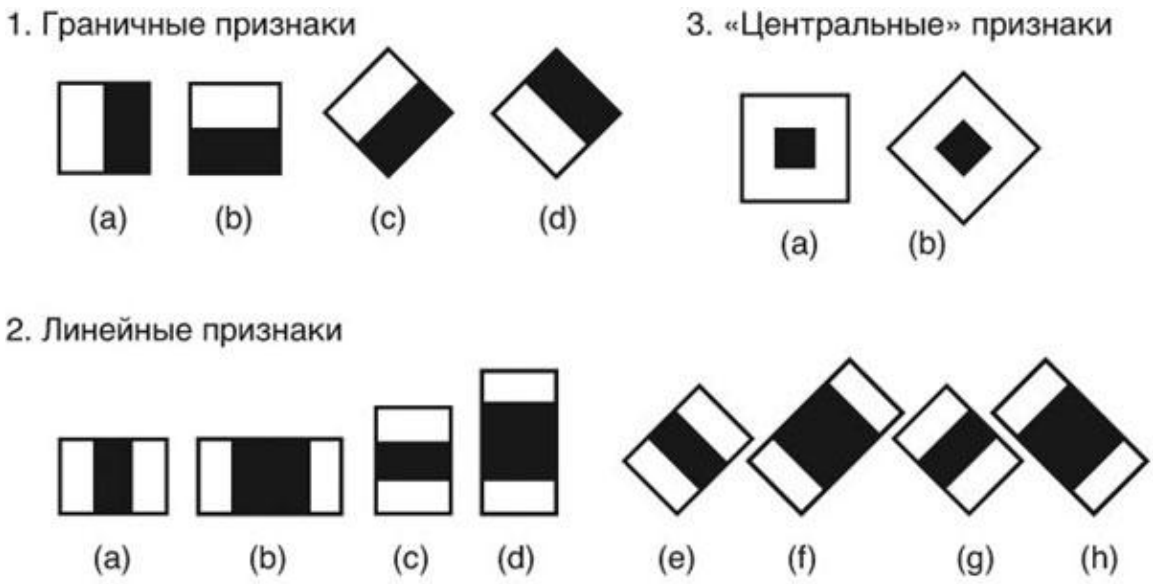
2.Результат работы

4.8. Распознавание лиц с использованием каскадов Хаара

Распознать лица на видео или фотографии очень просто, используя библиотеку OpenCV. Для начала, чтобы распознать лицо на изображении нужно найти, где само лицо расположено. Для этого мы должны выделить его основные компоненты, такие как нос, лоб, глаза, губы и т.д. Для решения данной задачи используются шаблоны, они же **признаки Хаара** на подобии таких:



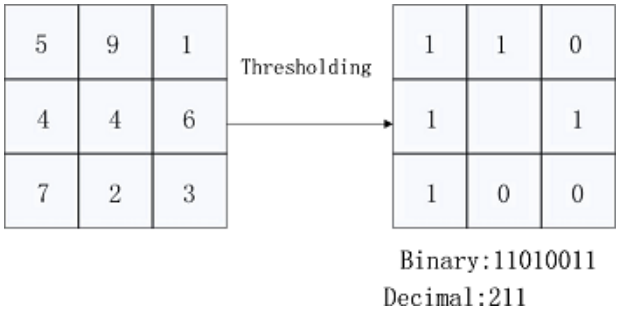
Признаки бывают разных видов:

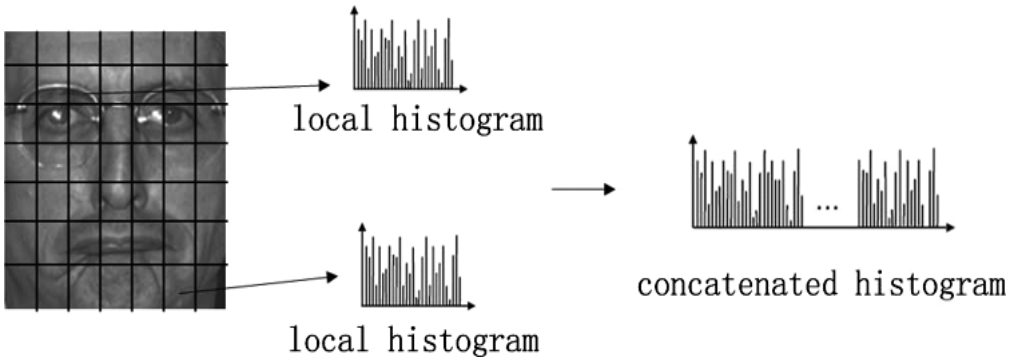


Если шаблоны соответствуют конкретным областям на изображении, можно считать, что на изображении есть человеческое лицо. На самом деле подобных шаблонов гораздо больше. Для каждого из них считается разность между яркостью белой и черной областей. Это значение сравнивается с эталоном и принимается решение о том, есть ли здесь часть человеческого лица или нет. Этот метод называется **методом Виолы-Джонса** (так же известен, как каскады Хаара). Однако, если на изображении не одно большое лицо, а много мелких, то лица не будут обнаружены, так как они будут меньше шаблонов. Для того чтобы искать на всем фото лица разных размеров используется метод скользящего окна. Именно внутри этого окна и высчитываются примитивы.

Метод скользящего окна — алгоритм трансформации, позволяющий сформировать из членов временного ряда набор данных, который может служить обучающим множеством для построения модели прогнозирования. Под окном в данном случае понимается временной интервал, содержащий набор значений, которые используются для формирования обучающего примера. В процессе работы алгоритма окно смещается по временной последовательности на единицу наблюдения, и каждое положение окна образует один пример. Окно как бы скользит по всему изображению. После каждого прохождения изображения окно увеличивается, чтобы найти лица большего масштаба.

После того, как лицо было обнаружено, нужно определить кому принадлежит это лицо. Для решения этой задачи используется алгоритм Local Binary Patterns. Суть его заключается в том, что мы разбиваем изображение на части и в каждой такой части каждый пиксель сравнивается с соседними 8 пикселями. Если значение центрального пикселя больше соседнего, то пишем 0, в противном случае 1. И так для каждого пикселя получается некоторое число. Далее на основе этих чисел для всех частей, на которые разбивали фотографию, считается гистограмма. Все гистограммы со всех частей объединяются в один вектор характеризующий изображение в целом. Если нужно узнать насколько похожи два лица, придется вычислить для каждого из них такой вектор и сравнить их. Рисунки ниже помогут лучше понять суть алгоритма:





С помощью описанных выше технологий, можно сделать полезный эффект размытия лиц, который используется, например, на телевидении для анонимных гостей.

Список используемых функций

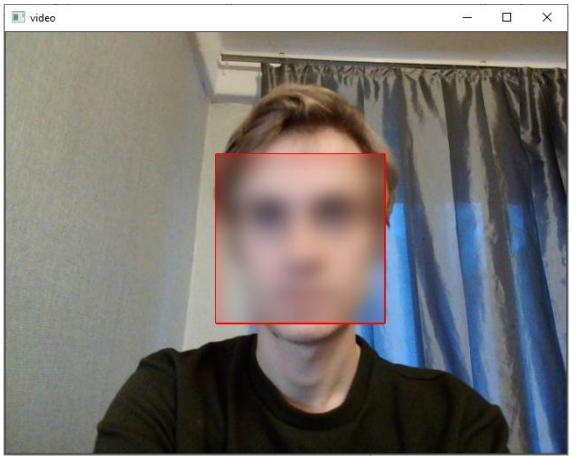
Название	Краткое описание
cv2.CascadeClassifier()	Метод для чтения модулей
cv2.GaussianBlur()	Метод для наложения Гауссовой фильтрации
cv2.namedWindow()	Задает название для окна
cv2.VideoCapture()	Метод определяет устройство воспроизведения видео. Начиная от 0 индексируются устройства. Если камера с индексом 0 занята, нужно поставить индекс 1
cap.read()	Считывает видео с камеры
detectMultiScale()	Метод определяет лица и возвращает массив
cv2.rectangle()	Строит прямоугольник в определенной области
cv2.imshow()	Выводит видео на экран
cv2.waitKey()	Метод ожидания нажатия клавиши. Возвращает код нажатой клавиши
cap.release()	Отпускаем устройство
cv2.destroyAllWindows()	Уничтожает все открытые окна

Листинг программы

```
import cv2

# Подключаем каскады Хаара
face_cascade_db = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")

# Создаем эффект размытия по Гауссу
def blur_face(img):
    (h, w) = img.shape[:2]
    dW = int(w / 3.0)
    dH = int(h / 3.0)
    if dW % 2 == 0:
        dW -= 1
    if dH % 2 == 0:
        dH -= 1
    return cv2.GaussianBlur(img, (dW, dH), 0)
```

```
if __name__ == '__main__':
    cv2.namedWindow("video")
    cap = cv2.VideoCapture(0)
    key = 0
    while (key != 27):
        # Считываем видео с камеры
        flag, img = cap.read()
        # С помощью функции detectMultiScale определяем лица и возвращает массив
        faces = face_cascade_db.detectMultiScale(img, scaleFactor=2,
minNeighbors=5, minSize=(30, 30))
        # Каждый элемент лежит в виде 4 позиций
        for (x, y, w, h) in faces:
            # Строим прямоугольник вокруг лица (необязательно)
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
            # Накладываем эффект именно для координат лица
            img[y:y + h, x:x + w] = blur_face(img[y:y + h, x:x + w])
        # Выводим результат
        cv2.imshow('video', img)
        key = cv2.waitKey(1)
    cap.release()
    cv2.destroyAllWindows()
```

Результат работы



4.9. Определение улыбки

Ниже представлена программа, которая работает по следующему алгоритму.

1. Подключение библиотек OpenCV и time.
2. Вывод на экран изображения с камеры
3. Конвертация изображения из цветовой палитры RGB в черно-белую палитру (для лучшего обнаружения контуров)
4. Обнаружение лица и отрисовка соответствующего прямоугольника на изображении

5. Обнаружение улыбки и отрисовка соответствующего прямоугольника на изображении
6. Если улыбка есть – увеличиваем счётчик времени
7. Если улыбки нет – обнуляем счётчик времени
8. Сохранение изображения в файл, если улыбка находится полторы секунды

Список использованных функций

Название функции	Описание
imwrite	Сохранение изображения в файл
cvtColor	Конвертация изображения из одной цветовой палитры в другую
rectangle	Отрисовка прямоугольника по координатам
imshow	Отображение изображения с камеры в окно программы
destroyAllWindows	Закрытие программы
waitKey	Ожидание ввода пользователя
VideoCapture	Захват видео с камеры

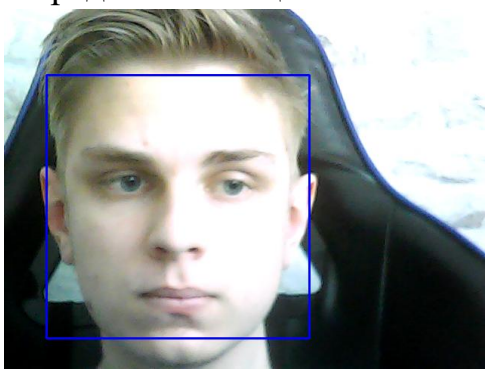
Листинг программы

```
import cv2
import time
if __name__ == '__main__':
    face_cascade =
cv2.CascadeClassifier('Resources/cascades/haarcascade_frontalface_default.xml')
    smile_cascade =
cv2.CascadeClassifier('Resources/cascades/haarcascade_smile.xml')
    video_capture = cv2.VideoCapture(0)
    cTime = 0
    while True:
        # получение картинки с камеры
        _, frame = video_capture.read()
        if cTime >= 1.5:
            # сохранение в файл, если улыбка держится полторы секунды
            cv2.imwrite("Saved pic.jpg", frame)
            cTime = 0
        # перевод картинки в чёрно-белый формат для лучшего определения
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # получение текущего времени
        pTime = time.time()
        # определение лица
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
        for (x, y, w, h) in faces:
            # отрисовка прямоугольника около лица
            cv2.rectangle(frame, (x, y), ((x + w), (y + h)), (255, 0, 0), 2)
            roi_gray = gray[y:y + h, x:x + w]
```

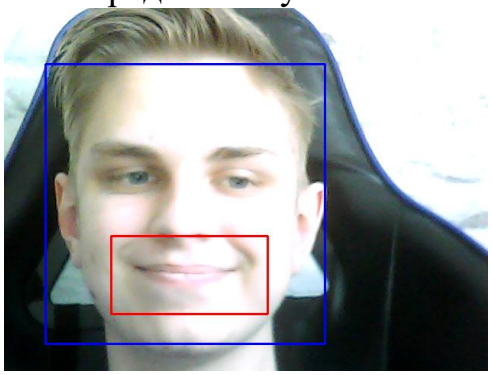
```
                roi_color = frame[y:y + h, x:x + w]
                smiles = smile_cascade.detectMultiScale(roi_gray, 1.3, 50)
                if len(smiles) != 0:
                    # если обнаружили
                    cTime = cTime + time.time() - pTime
                else:
                    cTime = 0
                for (sx, sy, sw, sh) in smiles:
                    # отрисовка прямоугольника около улыбки
                    cv2.rectangle(roi_color, (sx, sy), ((sx + sw), (sy + sh)), (0, 0,
255), 2)
            print(str(int(cTime)))
            cv2.imshow('Smile', frame)
            # если нажали Esc - выход
            if cv2.waitKey(1) == 27:
                break
        video_capture.release()
        cv2.destroyAllWindows()
        cv2.waitKey(0)
```

Скриншоты работы программ

Определение лица

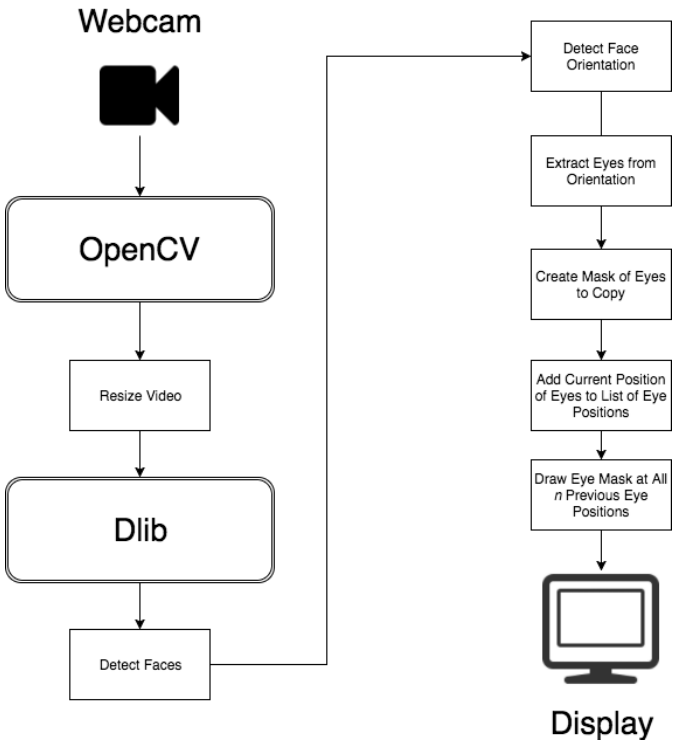


Определение улыбки



4.10. Распознавание и анимация глаз

В данном примере реализован алгоритм «эффекта линз».



Последовательность действий в алгоритме следующая.

Первоначально получаем видео с веб-камеры. Далее изменяем размер этого стрима, при помощи специальной функции imutils, чтобы получить нужную частоту кадров для определения лиц.

После получения частоты кадров, мы конвертируем кадр видео веб-камеры в черно-белый, затем передадим его Dlib для определения лица.

get_frontal_face_detector возвращает набор ограненных прямоугольников для каждого обнаруженного лица на изображении. Таким образом, мы можем использовать это как модель (в нашем случае, **shape_predictor_68_face_landmarks** на Github), и получить набор из 68-и точек с нашей лицевой ориентацией.

Из точек возле глаз мы можем **сформировать полигон** аналогичной формы в новом канале.

Таким образом, выполняем **bitwise_and** и копируем наши глаза из кадра.

Далее создаем объект для отслеживания n числа позиций, в которых были наши глаза. Функция **OpenCV** под названием **boundingRect** дает нам базовую координату x и y для рисования.

Наконец, мы создаем маску для воссоздания всех предыдущих местах, в которых были наши глаза, более того, **bitwise_and** копирует предыдущее изображение глаз в кадре перед показом.

Список основных используемых функций

Название	Описание
VideoStream().start()	Функция начинает запись
dlib.get_frontal_face_detector()	Функция обнаруживает наше лицо
dlib.shape_predictor()	Функция определяет расположение нашего лица
dlib.shape_predictor()[36:42]	Возвращает расположение левого глаза
dlib.shape_predictor()[42:48]	Возвращает расположение правого глаза
cv2.cvtColor()	Функция меняет цветовое пространство
vs.read()	Функция считывает видео с камеры
cv2.fillPoly()	Функция заливки маски
cv2.bitwise_and()	Функция вычисляет для каждого элемента побитовое соединение двух массивов или массива и скаляра.
cv2.boundingRect()	Функция используется для получения координат
cv2.waitKey()	Функция ожидания нажатия клавиши
cv2.imwrite()	Функция записи куда - либо
cv2.destroyAllWindows()	Функция уничтожает все открытые окна

Листинг программы

```
import argparse
import cv2
from imutils.video import VideoStream
from imutils import face_utils, translate, resize
import time
import dlib
import numpy as np

parser = argparse.ArgumentParser()
parser.add_argument("-predictor", required=True, help="path to predictor")
args = parser.parse_args()

print("starting program.")
```

```
print("'s' starts drawing eyes.")
print("'r' to toggle recording image, and 'q' to quit")

vs = VideoStream().start()
time.sleep(1.5)

# эта часть обнаруживает наше лицо
detector = dlib.get_frontal_face_detector()
# эта определяет расположение нашего лица
predictor = dlib.shape_predictor(args.predictor)

recording = False
counter = 0

class EyeList(object):
    def __init__(self, length):
        self.length = length
        self.eyes = []

    def push(self, newcoords):
        if len(self.eyes) < self.length:
            self.eyes.append(newcoords)
        else:
            self.eyes.pop(0)
            self.eyes.append(newcoords)

    def clear(self):
        self.eyes = []

# начинает от 10 предыдущих позиций глаз
eyeList = EyeList(10)
eyeSnake = False

# получаем первый кадр вне цикла, так что мы можем увидеть как
# вебкамера изменила свое расширение, которое теперь составляет w / np.shape
frame = vs.read()
frame = resize(frame, width=800)

frame = cv2.flip(frame, 1)

eyelayer = np.zeros(frame.shape, dtype='uint8')
eyemask = eyelayer.copy()
eyemask = cv2.cvtColor(eyemask, cv2.COLOR_BGR2GRAY)
translated = np.zeros(frame.shape, dtype='uint8')
translated_mask = eyemask.copy()

while True:
    # получаем кадр из камеры, уменьшаем размер
    frame = vs.read()
```

```
        frame = resize(frame, width=800)
        frame = cv2.flip(frame, 1)

        # заливаем наши маски и кадры 0 (черный цвет) в каждом цикле рисования
        eyelayer.fill(0)
        eyemask.fill(0)
        translated.fill(0)
        translated_mask.fill(0)

        # детектор ждет серые изображения
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        rects = detector(gray, 0)

        # для случая запуска цикла eyesnake (нажмите «s» во время работы для
активации)
        if eyeSnake:
            for rect in rects:
                # предсказатель это наша модель из 68 точек, которую мы загрузили
                shape = predictor(gray, rect)
                shape = face_utils.shape_to_np(shape)

                # наша модель dlib возвращает 68 точек, которые формируют лицо.
                # левый глаз расположен между 36й и 42й точками.
                # правый глаз – между 42й и 48й точками.
                leftEye = shape[36:42]
                rightEye = shape[42:48]

                # заливаем маску в форме наших глаз
                cv2.fillPoly(eyemask, [leftEye], 255)
                cv2.fillPoly(eyemask, [rightEye], 255)

                # копируем изображение из кадра в eyelayer при помощи этой маски
                eyelayer = cv2.bitwise_and(frame, frame, mask=eyemask)

                # это мы используем для получения координат x и y для вставки глаз
                x, y, w, h = cv2.boundingRect(eyemask)

                # добавляем это в наш список
                eyeList.push([x, y])

                # наконец, рисуем наши глаза в обратном порядке
                for i in reversed(eyeList.eyes):
                    # сначала меняем название eyelayer на eyes
                    translated1 = translate(eyelayer, i[0] - x, i[1] - y)
                    # далее, переводим присвоенную маску
                    translated1_mask = translate(eyemask, i[0] - x, i[1] - y)
                    # добавляем ее в существующую переведенную маску eyes (не
буквально, так как рискуем перегрузить)
                    translated_mask = np.maximum(translated_mask, translated1_mask)
```

```
                    # вырезаем новую переведенную маску
                    translated =  cv2.bitwise_and(translated,  translated,  mask=255  -
translated1_mask)
                    # вставляем в только-что указанную позицию глаза
                    translated += translated1
            # вырезаем переведенную маску еще раз
            frame = cv2.bitwise_and(frame, frame, mask=255 - translated_mask)
            # и вставляем в переведенное изображение глаза
            frame += translated

        # показ текущего кадра, проверяем, нажал ли пользователь на клавишу
        cv2.imshow("eye glitch", frame)
        key = cv2.waitKey(1) & 0xFF

        if recording:
            # создаем папку под названием "image_seq", теперь мы можем создавать гифки
            # в ffmpeg с последовательными изображениями
            cv2.imwrite("image_seq/%05d.png" % counter, frame)
            counter += 1
        if key == ord("q"):
            break
        if key == ord("s"):
            eyeSnake = not eyeSnake
            eyeList.clear()
        if key == ord("r"):
            recording = not recording

    cv2.destroyAllWindows()
    vs.stop()
```

Скриншот анимации

4.11. Детектирование оставленных предметов с использованием imutils и dnn

Для обеспечения антитеррористической безопасности в значительной части видеосистем, которые расположены в общественных местах большой популярностью пользуется детектор оставленных предметов.

Мы приведем реализацию детектирования объектов в реальном времени, для этого используем OpenCV, с помощью которого будет происходить вся работа с видео потоками.

Используем пакет **imutils**, разработанный Адрианом Роузброком, а именно классы этого пакета VideoStream и FPS. Также задействуем модуль **dnn** (deep learning), который предназначен для работы с нейронными сетями. Главной возможностью dnn является загрузка и запуск этих сетей. Мы будем пользоваться реализацией Caffe сети обнаружения SSD Google MobileNet.

Перечислим основные шаги, которые нужно проделать, чтобы реализовать детектор объектов в реальном времени:

1. Необходимо получить доступ к веб-камере.
2. Из видео потока получить кадр.
3. Преобразовать кадр в blob-объект. Blob (Binary Large Object) – объекты, которые хранят информацию в виде байтов.
4. Передать blob через сеть и получить обнаруженные объекты.
5. Для каждого объекта выполнить проверку на валидность (что схожесть более 25%).
6. Если проверка прошла, вычисляем центр нашего объекта, рисуем на кадре цветную рамку вокруг нашего объекта и подписываем его.

Использованные функции

Наименование функции	Краткое описание
cv2.dnn.readNetFromCaffe	Читает сетевую модель, хранящуюся в формате

	Caffe framework
VideoStream(src=0).start	Производит инициализацию видео потока веб-камерой и запускает его
FPS().start	Инициализирует fps по умолчанию и запускает таймер
vs.read	Возвращает текущий кадр
imutils.resize	Меняет размер кадра
cv2.dnn.blobFromImage	Конвертирует изображение в 4-мерный blob
net.setInput	Устанавливает blob-объект как входные данные в нашу сеть
net.forward	Передаёт входные данные через нейросеть, которая занимается обнаружением предметов.
cv2.putTex	Рисует текстовую строку на кадре
cv2.imshow	Отображает кадр в окне
cv2.waitKey	Ожидает нажатие клавиши
fps.update	Увеличивает общее количество проанализированных кадров
fps.stop	Останавливает таймер
fps.elapsed	Возвращает общее количество секунд между запуском и остановкой таймера
fps.fps	Вычисляет и возвращает количество обработанных кадров в секунду
cv2.destroyAllWindows	Закрывает все окна и освобождает память
vs.stop	Останавливает видео поток и освобождает все ресурсы

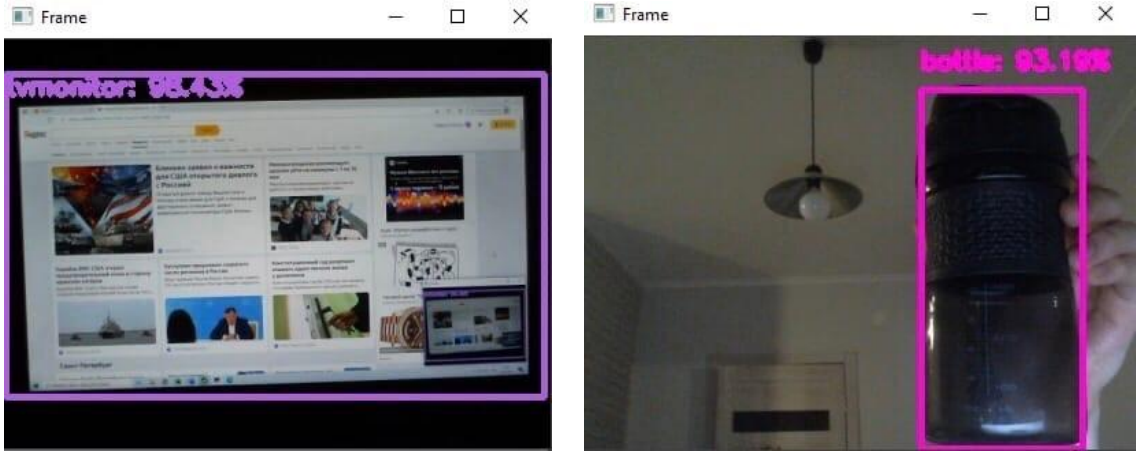
Листинг программы

```
# Выполняем импорт необходимых библиотек
import cv2
import argparse
import time
import numpy as np
import imutils
from imutils.video import VideoStream
from imutils.video import FPS

# Разбираем аргументы командной строки
argument = argparse.ArgumentParser()
argument.add_argument("-p", "--prototxt", required=True,
help="Specify the path to the required prototxt file")
argument.add_argument("-m", "--model", required=True,
help="Specify the path to the required pre-trained model")
argument.add_argument("-c", "--certainty", type=float, default=0.25,
help="Specify the minimum probability of similarity (default is 25%)")
args = vars(argument.parse_args())

# Инициализируем список меток классов
LABELS = ["aeroplane", "bird", "bicycle", "background", "boat",
```

```
    "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog",
    "horse", "motorbike", "person", "pottedplant", "sheep",
    "sofa", "train", "tvmonitor"]

    # Создаем список цветов ограничивающей рамки для каждого класса
    COLORS = np.random.uniform(0, 255, size=(len(LABELS), 3))

    # Читаем сетевую модель, хранящуюся в формате Caffe framework
    print("Wait, the model is being loaded")
    network = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

    # Выполняем инициализацию видео потока, в данном случае веб-камерой и запускаем
его
    print("Wait, the video stream is starting now")
    video_stream = VideoStream(src=0).start()
    time.sleep(3.0)

    # Инициализируем fps по умолчанию и запускаем таймер
    fps = FPS()
    fps.start()

    # Организуем бесконечный цикл в котором будем читать
    # кадры из видео потока и обрабатывать их
    while True:

    # Получаем текущий кадр из видео потока
    frame = video_stream.read()

    # Выполняем изменение размера кадра
    frame = imutils.resize(frame, width=400)

    # Получаем размеры кадра
    (h, w) = frame.shape[:2]

    # Преобразуем кадр в blob-объект
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
    0.007843, (300, 300), 127.5)

    # Передайте blob через сеть и получаем обнаруженные объекты
    network.setInput(blob)
    identifications = network.forward()

    # Организуем цикл по обнаруженным объектам
    for i in np.arange(0, identifications.shape[2]):

    # Получаем точность, связанную с предсказанием к какому классу относится объект
    certainty = identifications[0, 0, i, 2]

    # Проверяем что полученная точность больше чем заданные нами 25%
```

```
    if certainty > args["certainty"]:

    # Получаем индекс метки класса
    index = int(identifications[0, 0, i, 1])

    # Вычисляем координаты центра ограничивающего прямоугольника для распознанного
объекта
    rectangle = identifications[0, 0, i, 3:7] * np.array([w, h, w, h])
    (X1, Y1, X2, Y2) = rectangle.astype("int")

    # Получаем метку в виде строки
    label = "{}: {:.2f}%".format(LABELS[index],
    certainty * 100)

    # Рисуем ограничивающий прямоугольник
    cv2.rectangle(frame, (X1, Y1), (X2, Y2),
    COLORS[index], 3)

    # Вычисляем координату по Y для подписи метки
    y = Y1 - 15 if Y1 - 15 > 15 else Y1 + 15

    # Пишем на кадре метку
    cv2.putText(frame, label, (X1, y),
    cv2.FONT_HERSHEY_SIMPLEX, 0.6, COLORS[index], 3)

    # Отображает кадр в окне
    cv2.imshow("Frame", frame)

    # Ожидаем когда будет нажата определенная клавиша
    key = cv2.waitKey(1) & 0xFF

    # Если клавиша 's' нажата выходим из цикла
    if key == ord("s"):
    break

    # Увеличиваем общее количество всех проанализированных кадров
    fps.update()

    # Останавливаем таймер и печатаем информацию о времени и количестве кадров
обработанных в секунду
    fps.stop()
    print("The elapsed time: {:.2f}".format(fps.elapsed()))
    print("Approximate value of FPS: {:.2f}".format(fps.fps()))

    # Происходит закрытие всех окон и освобождение памяти
    cv2.destroyAllWindows()

    # Останавливаем видео поток и освобождаем все ресурсы
    video_stream.stop()
```

Результаты работы



4.12. Классификация изображений с использованием Inception V3

Сверточные нейронные сети — это тип нейронной сети с глубоким обучением. Эти нейронных сетей широко используются в компьютерном зрении и за последние несколько лет расширили возможности компьютерного зрения. Такое обучение позволяет повторно обучать последний слой существующей модели, что приводит к значительному сокращению не только времени обучения, но и размера необходимого набора данных. Возможность переобучить последний слой означает, что вы можете сохранить знания, полученные моделью во время первоначального обучения, и применить их к меньшему набору данных, что приведет к высокоточной классификации без необходимости обширного обучения и вычислительной мощности. Одна из самых известных моделей, которые можно использовать для трансферного обучения, - Inception V3.

Inception V3 – это сверточная нейронная сеть для помощи в анализе изображений и обнаружении объектов. Она помогает квалифицировать объекты в мире компьютерного зрения. Эта модель изначально была обучена более чем на миллионе изображений из 1000 классов на очень мощных машинах.

Описание используемого алгоритма

В этой работе будем квалифицировать медведей: бурый, панда, красная панда и полярный.

Для начала соберем фото и создадим две папки train и valid и заполним их. Получилось примерно 1000 фото каждого вида медведя в папке train и 200 фото в valid. Каждый вид должен находиться в разных папках.

Затем необходимо сделать аугментацию данных, будем брать изображения и производить над ними все необходимые трансформации. Загружаем картинки из каждой папки, затем прогоним эти изображения через обученную Inception V3 и сохраним получившуюся модель. В завершение, проверим ее работу.

Таблица используемых функций OpenCV

cv2.imread()	Загружает изображение из указанного файла
cv2.imshow()	Используется для отображения изображения в окне
cv2.waitKey()	Ждем до нажатия клавиши
cv2.destroyAllWindow()	Удаляем все открытые окна
cv2.putText()	Вставка изображения

Таблица используемых функций keras

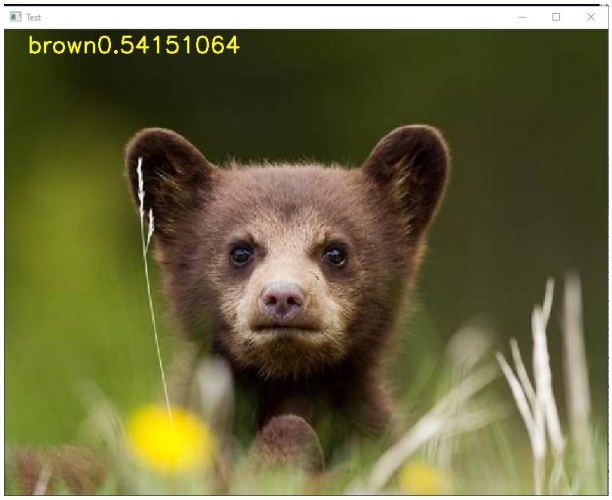
InceptionV3	Создание экземпляра InceptionV3
InceptionV3.layers	Извлекает один слой
InceptionV3.output	Извлекает тензор слоя
InceptionV3.output	Вставляет тензор слоя
Compile()	Обучение модели
Flow_from_directory	Получает изображения из директории
Fit_genetator	Модель данных, полученная от партии к партии
Save_weights	Сохранение модели
Load_weights	Загрузка модели
Predict()	Классификация

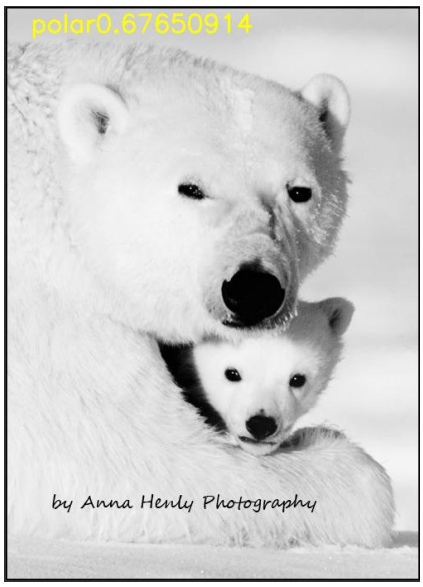
Листинг программ

```
     from keras import Model
from keras.layers import Dense
from keras.applications import InceptionV3
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import cv2

#создаем модель
def get_model(count_classes):
    inceptionV3 = InceptionV3('max')  # предобученная модель
    for layer in inceptionV3.layers:
        layer.trainable = False  # не обучаем предпобученную модель
    inceptionV3Out = inceptionV3.output
    output = Dense(count_classes, 'softmax')(inceptionV3Out)
    model = Model(inceptionV3.input, output)  # объявление модели
    model.compile('categorical_crossentropy', 'adam', ['accuracy'])
    return model
# тренировка
def train():
    trainDir = 'train'
    validationDir = 'val'
    trainGenerate = ImageDataGenerator(  # на основе известных изображений создает
похожие
        1. / 255,
        0.2,  # сдвиг на +-20%
        0.2,  # масштабирование на +-20%
        True)  # отражение по горизонтали
    testGenerate = ImageDataGenerator(1. / 255)
    batch_size = 64
    trainGenerator = trainGenerate.flow_from_directory(  # загружаем фото для
тренировки
        trainDir,  # директория с тренировочными данными
        (299, 299),  # целевой размер картинок
        batch_size,  # картинок за итерацию
        'categorical')

    validationGenerator = testGenerate.flow_from_directory(validationDir, (299, 299),
batch_size, 'categorical')
    countTrainImg = 3973
    countValImg = 673
    model = get_model(4)
    model.fit_generator(  # обучение модели
        trainGenerator, countTrainImg // batch_size, 20, validationGenerator,
countValImg // batch_size)
    model.save_weights('bears.h5')  # сохранение модели
def inference(file_name):
    img = np.array(image.load_img(file_name, target_size=(299, 299))) / 255.  # чтение
изображения из файла
```

```
    img = np.expand_dims(img, axis=0)
    result = model.predict(img)  # предсказание
    return result
model = get_model(4)
train()
pict = ['train/brown_bear/brown1.jpg', 'train/panda/panda1.jpg',
'train/polar_bear/polar.jpg',
        'train/red_pandas/red.jpg']  # объявляем для теста
bears = ["brown", "panda", "polar", "red"]
model.load_weights('bears.h5')
for k in range(4):
    res = inference(pict[k])
    print(res)
    t = 0
    y = bears[0]
    for i in range(4):
        if t < res[0][i]:
            t = res[0][i]
            y = bears[i]
    img = cv2.imread(pict[k])
    cv2.putText(img, y + (str)(t), (30, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
255), 2)
    cv2.imshow("Bear", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Скриншоты результатов выполнения программы

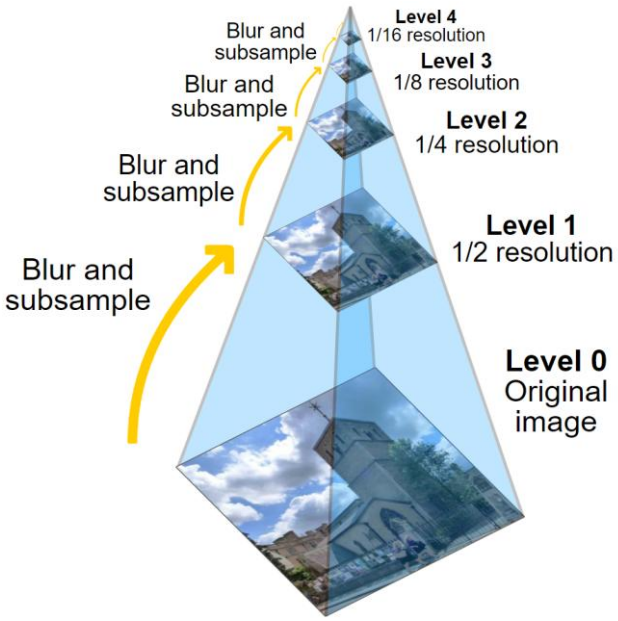




4.13. Пирамиды изображений

Пирамида изображения – это коллекция изображений, получаемая из исходного изображения путём последовательного сжатия, пока не будет достигнута точка останова. В данной программе были реализованы две пирамиды: «вверх» и «вниз».

В пирамиде «вниз» последующие изображения уменьшаются в масштабе. Каждый пиксель, содержащий локальное среднее значение, соответствует пикселю соседства на нижнем уровне пирамиды. Пример данной пирамиды (с 5-ю уровнями):



Реализованы пирамиды с помощью двух функций: pyrUp() и pyrDown(). Функция pyrUp() увеличивает размер в два раза по сравнению с его первоначальным размером, а pyrDown() уменьшает размер до половины. Если мы сохраним исходное изображение как базовое изображение и продолжим применять к pyrDown функцию pyrDown и будем хранить изображения в вертикальной стопке, оно будет выглядеть как пирамида (аналогично и для функции pyrUp).

Формат функций OpenCV:

pyrUp(src, dst, dstsize, borderType) – увеличивает размер изображения и размывает его и pyrDown(src, dst, dstsize, borderType) – уменьшает изображение и размывает его:

- src входное изображение
- dst выходное изображение. Он имеет указанный размер и тот же тип, что и src .
- dstsize размер выходного изображения
- borderType — переменная целочисленного типа, представляющая тип используемой границы.

Программа выполняет следующие действия:

Подключает 2 библиотеки: matplotlib и opencv.

Считывает изображения для последующих преобразований с ними, а также выводит исходные изображения.

В циклах также используется функция subplot, чтобы удобно создавать общие макеты подзаголовков в одном вызове.

Далее используются основные программы pyrUp и pyrDown и с помощью функции imshow отображаются изображения в окнах.

Использованные функции

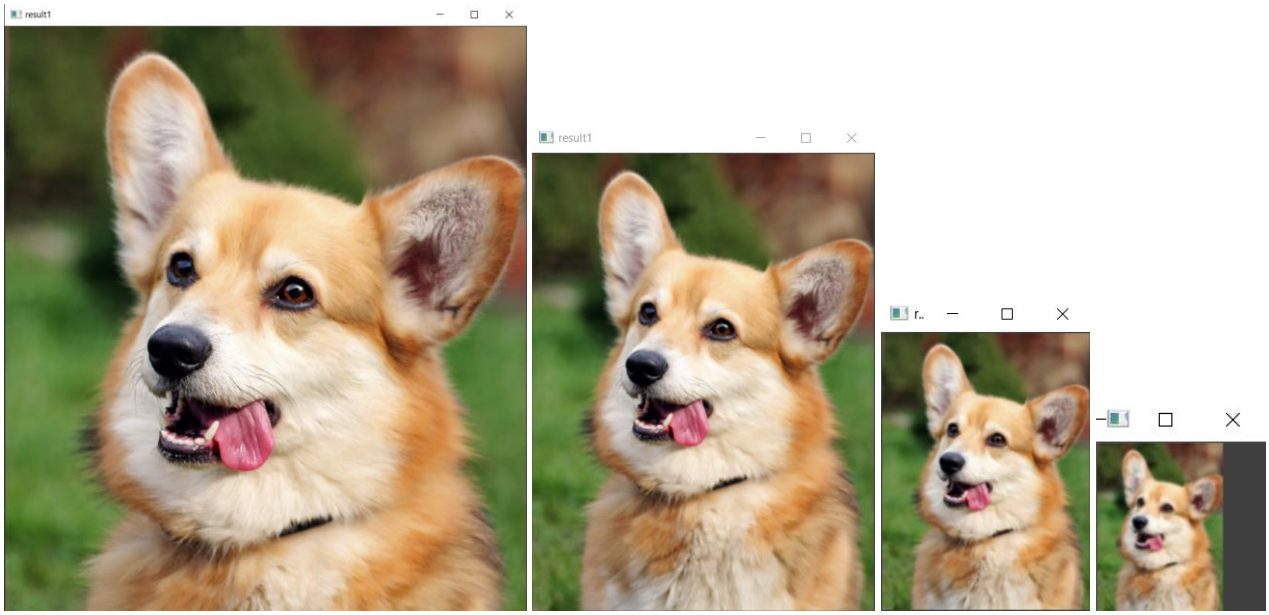
Функция	**Описание**
subplots(nrows=1, ncols=1, *, sharex=False, sharey=False,	Создаются фигура и набор подзаголовков. Данная функция позволяет удобно создавать

squeeze=True, subplot_kw=None, gridspec_kw=None, **fig_kw)	общие макеты подзаголовков в одном вызове.
pyrDown(src, dst=None, dstsize=None, borderType=None)	Размывает изображение и уменьшает его.
imshow()	Добавление и отображает изображения в окне. Принимает в себя два аргумента: название окна, в котором будет отрисовано изображение; имя переменной, которая хранит данное изображение.
waitKey(0)	Ждет нажатия клавиши. Данная команда останавливает выполнение скрипта до нажатия клавиши на клавиатуре. Параметр 0 означает, что нажатие любой клавиши будет засчитано.
destroyAllWindows()	Уничтожает все открытые окна и освобождает любое связанное с ним использование памяти.

Листинг программы

```
import cv2
import matplotlib.pyplot as plt

img1 = cv2.imread("corgi.jpg")
img2 = cv2.imread("m_corgi.jpg")

layer1 = img1.copy()
layer2 = img2.copy()

for i in range(4):
    plt.subplot(2, 2, i + 1)

    layer1 = cv2.pyrDown(layer1)
    plt.imshow(layer1)
    cv2.imshow("result1", layer1)
    cv2.waitKey(0)
cv2.destroyAllWindows()

for i in range(2):
    plt.subplot(2, 2, i + 1)

    layer2 = cv2.pyrUp(layer2)
    plt.imshow(layer2)
    cv2.imshow("result2", layer2)
    cv2.waitKey(0)
cv2.destroyAllWindows()
```

Результаты программы result1 – пирамида «вниз»:



4.14. Поиск прямых и окружностей на изображении с помощью схемы голосования. Преобразование Хафа

Преобразование Хафа (Hough Transform) — вычислительный алгоритм, применяемый для параметрической идентификации геометрических элементов растрового изображения. Предназначен для поиска объектов, принадлежащих определённому классу фигур, с использованием процедуры голосования. Процедура голосования применяется к пространству параметров, из которого и получаются объекты определённого класса фигур по локальному максимуму в так называемом накопительном пространстве (accumulator space), которое строится при вычислении трансформации Хафа.

Классический алгоритм преобразования Хафа связан с идентификацией прямых в изображении, но позже алгоритм был расширен возможностью идентификации позиции произвольной фигуры, чаще всего эллипсов и окружностей [39].

Преобразование Хафа для поиска прямых. Функции HoughLines и HoughLinesP

Прямая может быть представлена с помощью двух параметров. Например:

- с помощью параметров m и b: $y = mx + b$
- с помощью параметров θ и r : $y = \left(-\frac{cos\theta}{\sin\theta}\right)x + \left(\frac{r}{sin\theta}\right)$ (r— это длина радиус-вектора ближайшей к началу координат точки на прямой, а θ — это угол между этим вектором и осью абсцисс)

В преобразовании Хафа используются вторые два параметра.

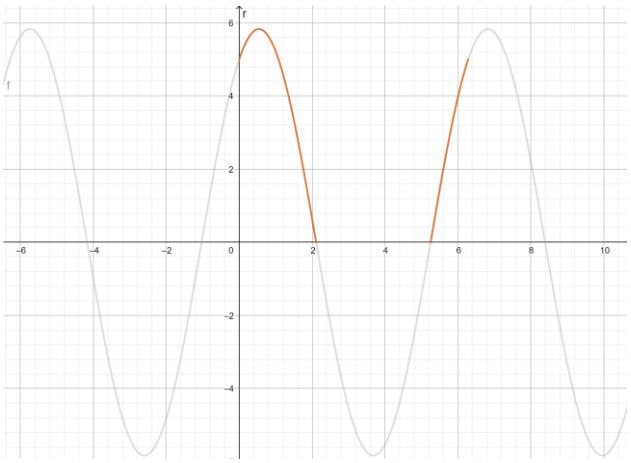
Преобразовав $y = \left(-\frac{cos\theta}{\sin\theta}\right)x + \left(\frac{r}{sin\theta}\right)$, получим $r = xcos\theta + ysin\theta$.

Для каждой точки (x_0, y_0) можно определить семейство прямых, проходящих через нее:

$r_\theta = x_0 \cdot cos\theta + y_0 \cdot sin\theta$

Каждая пара (r_θ, θ) определяет прямую, проходящую через (x_0, y_0).

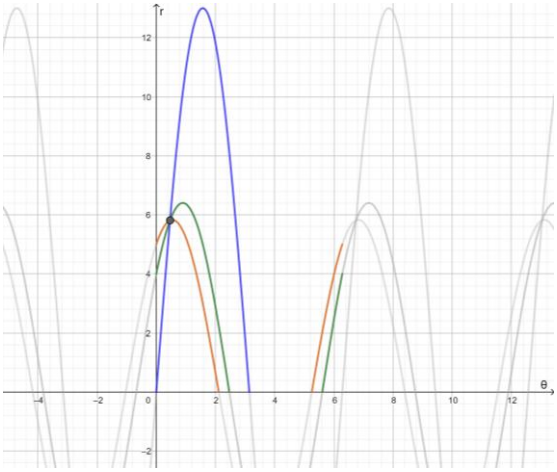
Если для заданных x_0 и y_0 построить в плоскости (θ, r) проходящее через точку (x_0, y_0) семейство прямых, то получим синусоиду. Например, при $x_0 = 5, y_0 = 3$:



Учитываем только точки, для которых $r > 0$ и $0 < \theta < 2\pi$

Такие синусоиды можно построить для всех точек изображения. Если для разных точек они пересекутся, это значит, что эти точки лежат на одной прямой. Например, для $x_1 = 5, y_1 = 3$,

$x_2 = 4, y_2 = 5$ и $x_3 = 0, y_3 = 13$, лежащих на прямой $y = -2x + 13$:



Это означает, что прямую можно найти путем поиска пересечений таких кривых. Чем больше их пересечется в одной точке, тем вероятнее, что перед нами прямая. Преобразование Хафа определяет количество пересечений кривых всех точек изображения, если это количество для каких-то r_θ и θ превышает некоторый порог, то считается, что обнаружена прямая с параметрами r_θ и θ.

OpenCV предоставляет две версии преобразования Хафа для поиска прямых.

HoughLines – стандартное преобразование Хафа

```
HoughLines(image, rho, theta, threshold[,
    lines[, srn[, stn[, min_theta[, max_theta]]]]])-> lines
```

image – исходное изображение
lines – вектор прямых на выходе, представленных парами (r, θ)
rho – разрешение по расстоянию в пикселях
theta – разрешение по углу (в радианах)
threshold – порог, который означает минимальное количество пересечений, после которого считается, что обнаружена прямая
srn, stn – параметры, используемые для многофункционального преобразования Хафа. Для использования стандартного преобразования выставляются равными 0
min_theta – минимальный угол для поиска прямых (должен находиться между 0 и max_theta)
max_theta - максимальный угол для поиска прямых (должен находиться между min_theta и π)

HoughLinesP – вероятностное преобразование Хафа (более эффективная реализация)

HoughLinesP(image, rho, theta, threshold[,
lines[, minLineLength[, maxLineGap]]]) -> lines

image – исходное изображение
lines – вектор прямых на выходе, представленных векторами (x_1, y_1, x_2, y_2) из 4-х элементов, где (x_1, y_1) и (x_2, y_2) – концы распознанного отрезка прямых
rho – разрешение по расстоянию в пикселях
theta – разрешение по углу (в радианах)
threshold - порог, который означает минимальное количество пересечений, после которого считается, что обнаружена прямая
minLineLength – минимальная длина отрезка прямой (отрезки меньшей длины отклоняются)
maxLineGap – максимальное расстояние между точками, лежащими на одной прямой, при котором их можно соединить

Преобразование Хафа для поиска окружностей

Преобразование Хафа для окружностей работает примерно аналогично преобразованию Хафа для прямых: окружности, в отличие от прямых, определяются тремя параметрами – $(x_{center}, y_{center}, r)$,

где (x_{center}, y_{center}) – центр окружности, r – ее радиус. Для эффективности OpenCV реализует модифицированную версию преобразования Хафа для поиска окружностей, которая сначала выделяет границы на изображении и находит возможные центры окружностей, а потом переходит ко второму этапу – нахождению наилучшего радиуса для каждого центра.

HoughCircles(image, method, dp, minDist[,
circles[, param1[, param2[,
minRadius[, maxRadius]]]]]) -> circles
image – исходное изображение
circles – вектор окружностей на выходе, представленных векторами $(x, y, radius)$ или $(x, y, radius, votes)$

method – метод распознавания (используем HOUGH_GRADIENT)
dp – Величина, обратная отношению разрешения накопителя к разрешению изображения
minDist – минимальное расстояние между двумя
param1 – назначение этого параметра зависит от метода. Для HOUGH_GRADIENT это наибольший порог из передаваемых в функцию Canny (внутри самой houghCircles)
param2 – назначение этого параметра зависит от метода. Для HOUGH_GRADIENT это порог для преобразования Хафа, используемый при поиске центров. Чем он меньше, тем больше ложных окружностей может быть обнаружено.
minRadius – минимальный радиус окружности
maxRadius – максимальный радиус окружности.

Алгоритм работы программы

1) Разбор аргументов командной строки с помощью модуля argparse

2) Чтение изображения из файла с предварительной проверкой на то, что его можно прочитать, и перевод его в черно-белую цветовую палитру

Дальнейшие действия зависят от того, какие объекты выбраны для распознавания.

Для прямых:

3) Применение функции HoughLines для получения вектора наборов параметров (r, θ)

4) Рисование отрезков прямых, полученных из HoughLines, с помощью функции line (точки получаются с помощью r и θ)

5) Применение функции HoughLinesP для получения вектора наборов параметров (x_1, y_1, x_2, y_2)

6) Рисование отрезков прямых, полученных из HoughLinesP, с помощью функции line

7) Вывод конечных и промежуточных результатов в отдельных окнах, если опция –no_show не использована

8) Запись результатов в файлы [args.output] и p_[args_output]

Для окружностей:

3) Применение функции medianBlur для уменьшения шума, чтобы избежать распознавания ложных окружностей

4) Применение функции HoughCircles для получения вектора наборов параметров $(x_{center}, y_{center}, r)$

5) Рисование окружностей и их центров с помощью функции circle

6) Вывод конечных и промежуточных результатов в отдельных окнах, если опция –no_show не использована

7) Запись результатов в выходной файл

Аргументы командной строки

Программа принимает два обязательных аргумента командной строки – название файла с изображением и объекты, которые нужно распознать – прямые (lines) или окружности (circles). Также как аргументы можно передать цвет выделения найденных объектов, имя файла для сохранения результата и опцию –no_show, позволяющую не выводить на экран промежуточные изображения и результаты преобразований.

Получение справки по аргументам командной строки с помощью -h:

Команда:

```
 .\hough.py -h
```

Вывод:

```
usage: hough.py [-h] [--no_show] [-o OUTPUT] [--color
{yellow,blue,lime,black,white,red,green,cyan,magenta,purple,gray}] filename
{circles,lines}

positional arguments:
  filename              The name of the file containing an image
  {circles,lines}       The shapes that need to be detected

optional arguments:
  -h, --help            show this help message and exit
  --no_show             Prevents the program from showing processing results in
                        windows
  -o OUTPUT, --output OUTPUT
                        The file to write the results to. Note that in the case of
                        lines detection the filename will be prepended with 'p_' for
                        the probabilistic line transform
  --color {yellow,blue,lime,black,white,red,green,cyan,magenta,purple,gray}, -c
{yellow,blue,lime,black,white,red,green,cyan,magenta,purple,gray}
                        The color to highlight shapes with
```

Таблица использованных функций

Название функции	Описание
imread	Загрузка изображения из файла

cvtColor	Конвертация изображения из одной цветовой палитры в другую
Canny	Функция обнаружения краев
resize	Функция изменения размера (ширины и высоты) изображения
HoughLines	Стандартное преобразование Хафа для поиска прямых
HoughLinesP	Вероятностное преобразование Хафа для поиска прямых
HoughCircles	Преобразование Хафа для поиска окружностей
line	Функция рисования отрезка
circle	Функция рисования окружности
medianBlur	Функция размытия изображения с помощью медианного фильтра
imwrite	Запись изображения в файл
imshow	Вывод изображения на экран
waitKey	Функция ожидания нажатия клавиши
haveImageReader	Функция, определяющая, может ли изображение быть расшифровано OpenCV

Листинг программы

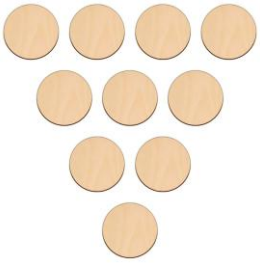
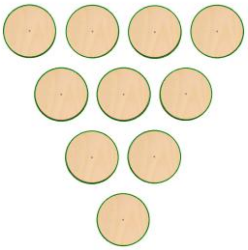
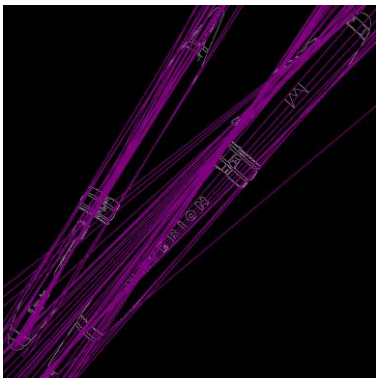
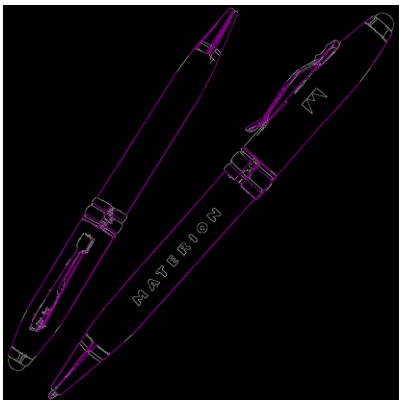
```
import argparse
import numpy as np
import math
import cv2 as cv
import sys

def downsize(image):
    k = 0.5
    x = int(k * image.shape[1])
    y = int(k * image.shape[0])
    return cv.resize(image, (x, y), k, k)


COLORS = {"yellow": (0, 255, 255),
          "blue": (255, 0, 0),
          "lime": (0, 255, 0),
          "black": (0, 0, 0),
          "white": (255, 255, 255),
          "red": (0, 0, 255),
          "green": (0, 128, 0),
          "cyan": (255, 255, 0),
          "magenta": (255, 0, 255),
          "purple": (128, 0, 128),
          "gray": (128, 128, 128)
          }

THICKNESS = 2
```

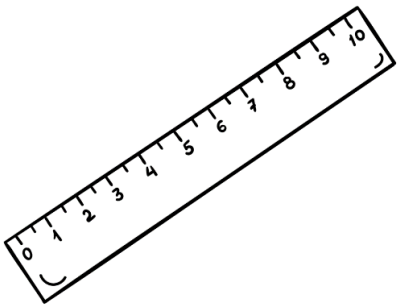
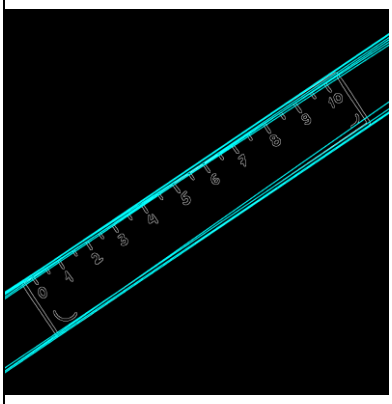
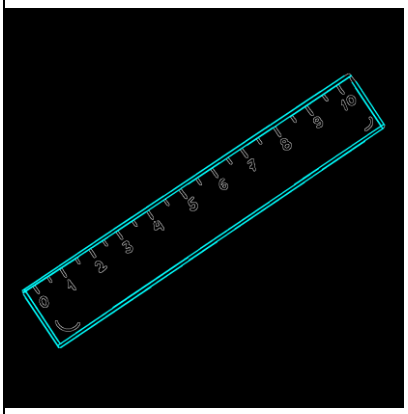
```
parser = argparse.ArgumentParser()
parser.add_argument('filename', help='The name of the file containing an image')
parser.add_argument('shape', help='The shapes that need to be detected',
                    choices=('circles', 'lines'))
parser.add_argument('--no_show', help='Prevents the program from showing'
                                      ' processing results in windows',
                    action='store_true')
parser.add_argument('-o', '--output', help='The file to write the results to.'
                                           ' Note that in the case of lines'
                                           ' detection the filename will be'
                                           ' prepended with \'p_\' for the'
                                           ' probabilistic line transform',
                    default='output.png')
parser.add_argument('--color', '-c', help='The color to highlight shapes with',
                    choices=COLORS.keys(), default="magenta")
args = parser.parse_args()

detecting_lines = (args.shape == "lines")
color = COLORS[args.color]

if not cv.haveImageReader(args.filename):
    print("Could not read the image from file")
    sys.exit(1)

src = cv.imread(cv.samples.findFile(args.filename), cv.IMREAD_COLOR)
gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)

if detecting_lines:
    dst = cv.Canny(gray, 50, 200, None, 3)
    cdst = cv.cvtColor(dst, cv.COLOR_GRAY2BGR)
    cdstP = np.copy(cdst)

    lines = cv.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)
    if lines is not None:
        for line in lines:
            rho = line[0][0]
            theta = line[0][1]
            a = math.cos(theta)
            b = math.sin(theta)
            x0 = a * rho
            y0 = b * rho
            pt1 = (int(x0 + 900 * (-b)), int(y0 + 900 * a))
            pt2 = (int(x0 - 900 * (-b)), int(y0 - 900 * a))
            cv.line(cdst, pt1, pt2, color, THICKNESS, cv.FILLED)

    linesP = cv.HoughLinesP(dst, 1, np.pi / 180, 50, None, 50, 10)

    if linesP is not None:
        for line in linesP:
```

```
            l = line[0]
            cv.line(cdstP, (l[0], l[1]), (l[2], l[3]), color,
                    THICKNESS, cv.FILLED)

    if not args.no_show:
        cv.imshow("Source", downsize(src))
        cv.imshow("Canny Edge Detection", downsize(dst))
        cv.imshow("Grayscale", downsize(gray))
        cv.imshow("Standard Hough Line Transform",
                  downsize(cdst))
        cv.imshow("Probabilistic Hough Line Transform",
                  downsize(cdstP))

    cv.imwrite(args.output, cdst)
    cv.imwrite('p_' + args.output, cdstP)

else:
    gray = cv.medianBlur(gray, 7)
    height = gray.shape[0]
    circles = cv.HoughCircles(gray, cv.HOUGH_GRADIENT, 1, height / 8,
                              param1=100, param2=35,
                              minRadius=1,
                              maxRadius=int(0.5 * min(gray.shape[:2])))

    if circles is not None:
        circles = np.uint16(np.around(circles))
        for circle in circles[0, :]:
            center = tuple(circle[:2])
            cv.circle(src, center, 1, COLORS["gray"], 3)
            radius = circle[2]
            cv.circle(src, center, radius, color, THICKNESS)

    if not args.no_show:
        cv.imshow("Median blur", downsize(gray))
        cv.imshow("The Circles Detected", downsize(src))
    cv.imwrite(args.output, src)

cv.waitKey(0)
```

Вывод программы

Входное изображение	Команда	Выходное изображение

	.\hough.py tennis_balls.jpg circles -c red	
	.\hough.py wood.webp circles -c green	
	.\hough.py pens.jfif lines --color purple	Стандартное преобразование  Вероятностное преобразование 

	.\hough.py ruler.png lines --color cyan	Стандартное преобразование  Вероятностное преобразование 

4.15. Создание панорамных снимков

Для построения панорамного изображения необходимо использовать следующие методы компьютерного зрения и обработки изображения (openCV):

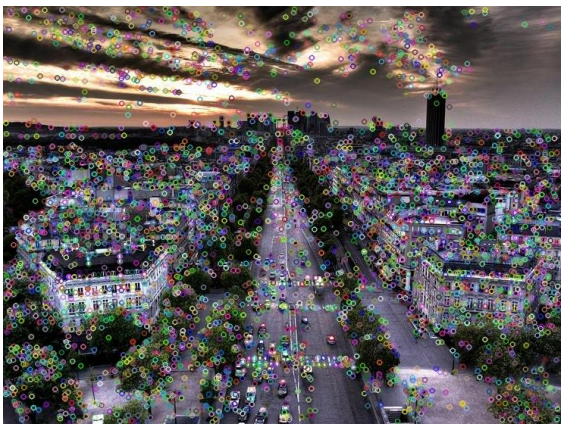
- Обнаружение ключевых точек;
- Нахождение локальных инвариантных дескрипторов;
- RANSAC – стабильный метод оценки параметров модели на основе случайных выборок. Так как данные могут быть сильно зашумлены, стандартный метод наименьших квадратов может привести к тому, что модель, полученная данным методом, может быть неточной или вовсе неверной. RANSAC же в свою очередь берет две точки, строит между ними прямую, тем самым образуя модель, которая уже проверяется

на предмет того, сколько точек ей соответствует, лучший результат и будет корректной моделью.

• Искривление перспективы.

Создание панорамного снимка может быть разделено на несколько этапов:

1. Обнаружение всех ключевых точек и инвариантных дескрипторов изображений при помощи метода detectAndCompute() абстрактного базового класса для 2D изображений, результат работы данного метода можно видеть на рисунках ниже, где первый рисунок – исходное изображение, второй – изображение с наложением всех найденных ключевых точек:



2. Сопоставление дескрипторов между двумя изображениями, для этого необходимо создать объект (здесь и далее matcher) для сопоставления описателей ключевых точек при помощи метода DescriptorMatcher_create() библиотеки openCV, где в качестве аргумента необходимо передать наименование необходимого matcher.

После создания нужного matcher необходимо вызвать у данного объекта метод knnMatch, где в качестве аргументов передать дескрипторы двух изображений. Данный метод найдет n лучших совпадений для каждого дескриптора в заданном списке.

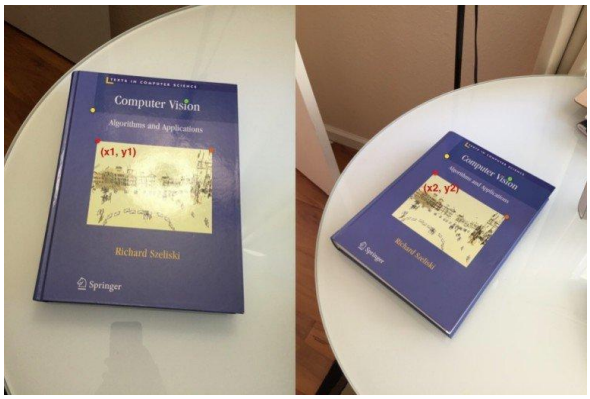
Возьмем 2 заранее разделенных изображений:

Применим вышеописанный метод сопоставления дескрипторов и получим следующий результат:

Данный метод нашел самые лучшие совпадения между изображениями (точки соответствия).

3. Использование алгоритма RANSAC для оценки матрицы гомографии. Допустим, у нас имеется два изображения одного и того же объекта под разными углами. При помощи предыдущего алгоритма мы нашли точки соответствий. Гомография же представляет собой матрицу размерностью 3 X 3, которая отображает точки одного изображения в точки соответствия другого изображения.

На картинке ниже можно выделить несколько точек соответствий, отмеченных своим цветом (соответствующие друг другу выделены одним цветом):



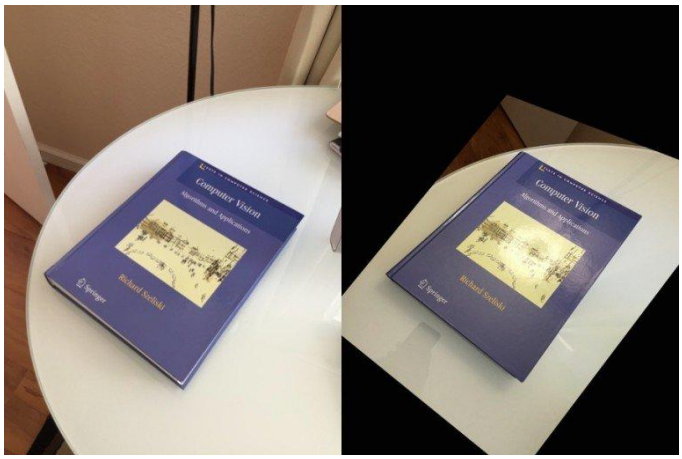
Так как гомография является матрицей 3 на 3, ее можно представить следующим образом:

$$\mathbf{H} = \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix}$$

Рассмотри первый набор точек соответствия – (x1, y1) в первом изображении и (x2, y2) во втором. Далее гомография отображает их следующим образом:

$$\begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = \mathbf{H} \times \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix}$$

Данное уравнение будет корректным для всех наборов точек соответствий, важным условием является лишь то, что эти точки должны лежать в одной плоскости. Исходя из этого, если применить гомографию, то второе изображение можно будет изменить таким образом, что точки соответствия будут иметь одинаковые координаты, следовательно и само изображение преобразуется таким образом, что будет таким же, как и первое, пример подобного преобразования показан на рисунке ниже.



Для нашей программы необходимо воспользоваться методом findHomography библиотеки OpenCV, где в качестве аргументов необходимо передать точки соответствий, идентификатор необходимого алгоритма (cv2.RANSAC).

4. Использовать преобразование перспективы на основе найденной нами матрицы гомографии при помощи метода warpPerspective() библиотеки OpenCV, где в качестве аргументов передаем второе изображение, матрицу гомографии и размер результирующей картинки.

5. Последним шагом в создании панорамного снимка является обычная склейка первой картинки и второй, полученной из 4 шага после преобразования перспективы. Ниже приведены примеры созданных панорамных изображений.

Как видно из последнего изображения, при создании панорамного снимка, возможны появления графических артефактов в виде швов состыковки. Подобные артефакты связаны с тем, что изначально эти два изображения могли быть сделаны под разным освещением и алгоритмами обработки изображений аппаратурой, при помощи которой были сделаны снимки. Решением данной проблемы возможно было бы создание плавного перехода одного цвета в другой в районе шва соединения, наложение градиентного перехода или цветовой коррекции самих изображений.

Список использованных функций

imread()	Чтение изображения из файла
imshow()	Отрисовка изображения в отдельном окне
imwrite()	Запись изображения в файл
cvtColor()	Изменение цветовой палитры изображения
xfeatures2d.SIFT_create()	Создание SIFT объекта
detectAndCompute()	Вычисление ключевых точек и дескрипторов при помощи определенного

	алгоритма (в нашем случае алгоритм SIFT)
DescriptorMatcher_create()	Создание сопоставителя ключевых точек для нахождения точек соответствий
knnMatch()	Поиск точек соответствий при помощи алгоритма KNN
findHomography()	Вычисление матрицы гомографии по точка соответствий
warpPerspective()	Изменение перспективы изображения в соответствии с матрицей гомографии
line()	Отрисовка на изображении линий

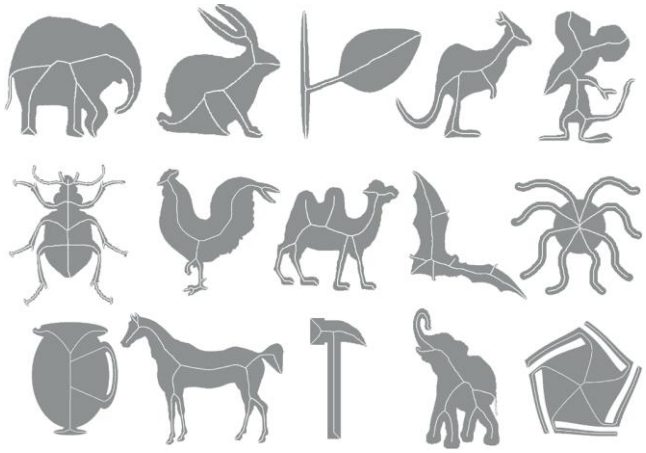
Листинг программы

```
import cv2
import numpy as np

def detectAndDescribe(image, method=None):

    # Создание черно-белого изображения
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Создание анализатора изображения для поиска ключевых точек по заданному методу
    if method == 'sift':
        descriptor = cv2.xfeatures2d.SIFT_create()
    elif method == 'surf':
        descriptor = cv2.xfeatures2d.SURF_create()
    elif method == 'brisk':
        descriptor = cv2.BRISK_create()
    elif method == 'orb':
        descriptor = cv2.ORB_create()

    # Получение ключевых точек и дескрипторов
    (keyPoints, features) = descriptor.detectAndCompute(gray, None)

    keyPoints = np.float32([kp.pt for kp in keyPoints])

    return (keyPoints, features)

def matchKeypoints(keyPointsA, keyPointsB, featuresA, featuresB,
                   ratio, reprojThresh):
        # Генерация сопоставителя
        matcher = cv2.DescriptorMatcher_create("BruteForce")
        # Поиск точек соответствий
        rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
        matches = []
        for m in rawMatches:
            if len(m) == 2 and m[0].distance < m[1].distance * ratio:
                matches.append((m[0].trainIdx, m[0].queryIdx))

        if len(matches) > 4:
            # Составление последовательности точек
            pointsA = np.float32([keyPointsA[i] for (_, i) in matches])
            pointsB = np.float32([keyPointsB[i] for (i, _) in matches])
            # Нахождение матрицы гомографии
            (H, status) = cv2.findHomography(pointsB, pointsA, cv2.RANSAC,
                                             reprojThresh)
            return (matches, H, status)
        return None
```

```
def stitch(images, ratio=0.75, reprojThresh=4.0, showMatches=False, method="sift"):
    # Распаковка изображений
    (imageA, imageB) = images
    (keyPointsA, featuresA) = detectAndDescribe(imageA, method)
    (keyPointsB, featuresB) = detectAndDescribe(imageB, method)

    # Поиск точек соответствий
    M = matchKeypoints(keyPointsA, keyPointsB, featuresA, featuresB, ratio,
reprojThresh)

    if M is None:
        return None

    (matches, H, status) = M
    # Преобразование перспективы для второго изображения и создание панорамного снимка
    result = cv2.warpPerspective(imageB, H, (imageA.shape[1] + imageB.shape[1],
imageA.shape[0] if (imageA.shape[0] > imageB.shape[0]) else imageB.shape[0]))
    result[0:imageA.shape[0], 0:imageA.shape[1]] = imageA

    if showMatches:
        # Получения изображение с наложением всех совпадений
        vis = drawMatches(imageA, imageB, keyPointsA, keyPointsB, matches, status)
        return (result, vis)
    return result


def drawMatches(imageA, imageB, keyPointsA, keyPointsB, matches, status):
    # Инициализация выходного изображения
    (hA, wA) = imageA.shape[:2]
    (hB, wB) = imageB.shape[:2]
    visualization = np.zeros((max(hA, hB), wA + wB, 3), dtype="uint8")
    visualization[0:hA, 0:wA] = imageA
    visualization[0:hB, wA:] = imageB
    # цикл по всем совпадениям
    for ((trainIdx, queryIdx), s) in zip(matches, status):
        # выполнение в случае совпадения ключевой точки
        if s == 1:
            # отрисовка совпадения
            pointA = (int(keyPointsA[queryIdx][0]), int(keyPointsA[queryIdx][1]))
            pointB = (int(keyPointsB[trainIdx][0]) + wA, int(keyPointsB[trainIdx][1]))
            cv2.line(visualization, pointA, pointB, (0, 255, 0), 1)
    # возвращение изображения с наложенными линиями между точек соответствий
    return visualization

if __name__ == '__main__':
    # Загрузка исходных изображений
    imageA = cv2.imread("C:\\Users\\serge\\Downloads\\MiniCity1.jpg")
    imageB = cv2.imread("C:\\Users\\serge\\Downloads\\MiniCity2.jpg")

    # Вызов метода для создание панорамного снимка
    (result, visualization) = stitch((imageA, imageB), showMatches=True)
    # Отрисовка панорамного изображения
    cv2.imshow('Result image', result)
    # Запись в файл полученных изображений
    cv2.imwrite('City3.jpg', result)
    cv2.imwrite('CityTest.jpg', visualization)
    # Отрисовка изображения со всеми найденными соответствиями
    cv2.imshow('Image matches', visualization)
    cv2.waitKey(0)
```

4.16. Скелетонизация

Скелетонизация это процесс изменения бинарного изображения в скелет фигуры – единую линию, образованную множеством точек, равноудаленных от границ изначальной фигуры.



Скелетонизация часто используется в обработке цифровых изображений, в частности их распознавании, отслеживании, сжатии, сравнении и так далее. Есть несколько методов скелетонизации: Сужение, Построение Диаграммы Вороного и Преобразование расстояния. В примере ниже использован метод сужения со следующим алгоритмом:

Алгоритм работы:

1) Подключение библиотеки OpenCV
2) Принимаем исходное черно-белое изображение
3) Создаем из него бинарное изображение
4) Создаем элемент при помощи getStructuringElement, который понадобится для сужения
5) Входим в бесконечный цикл, в котором:

5.1. Наложение формы на изображение

5.2. Вычитаем из оригинала изображения с наложением

5.3. Сужение изображения-результата

5.4. Совершаем побитовое или с результатом предыдущей итерации (на первой итерации просто задаем изображение полученное на первом этапе)

5.5. Сохраняем суженый результат для последующего сужения и повторяем цикл

6) Выходим из цикла когда не остается линий для сужения

7) Выводим результат

Список использованных функций

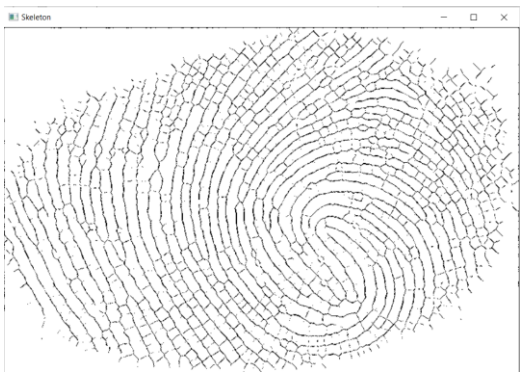
Название функции	Описание
imread	Считывание исходного изображения
threshold	Создает бинарное изображение из исходного черно-белого изображения, передаем параметры (img, 127, 255, 0) Все цвета со значениями больше 127 (2 параметр) станут белыми(255), а все с меньшим черными(0)
zeros	Возвращает массив нулей определенной формы (img.shape), в который мы запишем скелет
getStructuringElement	Возвращает структурирующий элемент определенный формы: элементы помогают «фильтровать» 2D изображения, поэтапно просматривая его, получая доступные для морфологического преобразования окрестности.
morphologyEx	Наложение структурирующего элемента и исполняем сужение с расширением, избавляясь от точечного шума.
subtract	Побитовое вычитание
erode	Функция сужения, которая и дает нам скелет изображения
bitwise_or	Побитовое или
copy	Копирование изображения
countNonZero	Считает ненулевые пиксели, то есть нечерные. Используем для выхода из цикла чтобы проверить остались ли возможности для сужения
bitwise_not	Побитовое нет. Используем для более наглядного вывода изображения-итога
imshow	Показ результата

Листинг программы

```
import cv2
import numpy

# Принимаем изображение, черно-белое.
original = cv2.imread('D://test3.jpg', 0)
```

```
    # Threshold команда создающая бинарное изображение из исходного черно-белого изображения Работает по принципу того
    # что все цвета со значениями больше 127 (2 параметр) станут белыми(255), а все с меньшим черными(0)
    ret,original = cv2.threshold(original, 127, 255, 0)
    # Ориентируясь по размеру исходного изображения создаем скелет, пока что пустой
    size = numpy.size(original)
    skeleton = numpy.zeros(original.shape, numpy.uint8)

    # Создание необходимой для операции erode формы, матрица 3 на 3, Выбрана форма DILATE по причине того что в ходе
    # экспериментов с формами она дала самый чистый результат
    structuringElement = cv2.getStructuringElement(cv2.MORPH_DILATE, (3,3))
    while True:
        #Наложение нашей формы на изображение
        cleared = cv2.morphologyEx(original, cv2.MORPH_OPEN, structuringElement)
        #Вычитание из оригинала наложенной фигуры
        subtracted = cv2.subtract(original, cleared)
        #применение функции Erode, которая "сужает" наш силуэт.
        eroded = cv2.erode(original, structuringElement)
        #побитовое или между пустым скелетом (в следующих итерациях не пустым) и полученным изображением
        skeleton = cv2.bitwise_or(skeleton,subtracted)
        #Сохраняем суженный результат для дальнейшего сужения.
        original = eroded.copy()
        # Мы сужаем белые пиксели, когда не остается достаточных для сужения линий - мы выходим из цикла.
        if cv2.countNonZero(original) == 0:
            break
    cv2.imshow("Skeleton",cv2.bitwise_not(skeleton))
    cv2.imshow("Original",cv2.imread('D://test3.jpg', 0))
    cv2.waitKey(0)
```

Пример работы



4.17. Детекторы областей IBR, MSER

На большинстве изображений есть области, которые могут быть обнаружены с высокой повторяемостью, поскольку они обладают

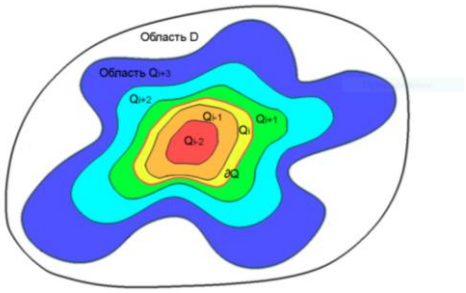
некоторыми отличительными, инвариантными и стабильными свойствами. Такие особые области или DR (distinguished regions), могут служить элементами, по которым можно выявлять соответствия при сопоставлении изображений. Одним из методов обнаружения этих локальных особенностей на изображение является метод «Максимально Стабильных Экстремальных Областей» или MSER (Maximally stable extremal regions). Этот метод устойчив к изменениям масштаба, вращениям и освещения. Он использовался для распознавания объектов, поиска изображений, обнаружения текста.

Метод MSER официально был представлен в 2002 году, на конференции по компьютерному зрению «British Machine Vision». Доклад представлял профессор чешского технического университета Иржи Матас, совместно с О. Хумом, М. Урбаном и Т. Паидлом. В последующие годы над методом продолжали работать, появлялись новые модификации и расширения. В 2011–2013 годах было опубликовано несколько статей в области обнаружения и распознавания текста с использованием метода MSER. Алгоритм MSER актуален по сей день, находятся новые области его применения и продолжаются исследования.

MSER – тип элементов изображения пригодных для нахождения соответствий. Элементы определяются экстремальным свойством функции интенсивности в области и на ее внешней границе.

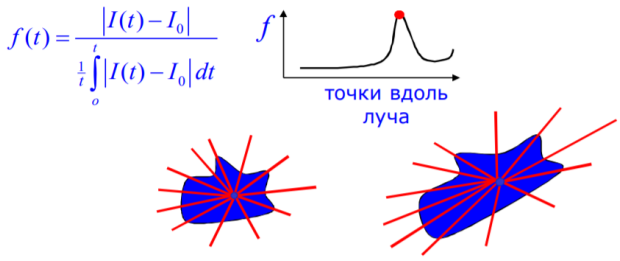
Перечисление экстремальных регионов происходит следующим образом. Сначала пиксели сортируются по интенсивности. Если диапазон значений изображения мал, то сортировка может быть реализована как блочная. После сортировки пиксели помещаются в изображение (либо в порядке убывания, либо в порядке возрастания), а список связных компонент и их площадей поддерживается с использованием алгоритма

для систем непересекающихся множеств. В процессе создается структура данных, сохраняющая площадь каждого связного компонент. Слияние двух компонентов рассматривается как прекращение существования меньшего компонента, и вставка всех пикселей меньшего компонента в более крупный. Уровни интенсивности, являющиеся локальными минимумами скорости изменения функции площади, выбираются как пороги, создающие максимально устойчивые экстремальные области. Каждая экстремальная область является связной компонентой порогового изображения. На выходе каждая максимально стабильная экстремальная область представлена расположением локального минимума или максимума интенсивности и порога. При обнаружении MSER, ищется ряд пороговых значений, которые практически не изменяются по отношению друг к другу. Для каждого множества вложенных экстремальных областей эти пороговые значения могут быть различны. Для некоторых частей изображения может существовать несколько устойчивых порогов.



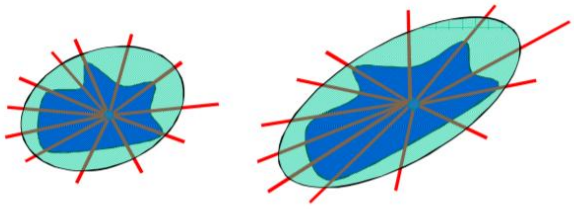
Максимально стабильные экстремальные области

Детектор областей IBR. Суть алгоритма заключается в том, чтобы идти от локального экстремума яркости по лучам, считая некоторую величину f. Остановка происходит при достижении пика f.



Детектор областей IBR.

Области на паре изображений могут различаться, поэтому опишем вокруг них эллипсы. Если эллипсы превратить в окружности, то получим полное сходство изображений с точностью до поворота.



Эллипсы, описанные вокруг областей

Таблица использованных функций

Image.open()	Загружает изображение из указанного файла. Возвращает объект типа класса PIL. PIL (Python Imaging Library)— библиотека для открытия, работы и сохранения различных форматов изображений.
cv2.imread()	Метод для чтения изображения. Возвращает 2D или 3D матрицу (в зависимости от количества цветовых каналов). Если изображение не может быть прочитано, возвращает пустую матрицу.
cv2.cvtColor()	Метод для преобразования изображения из одного цветового пространства в другое, с параметром cv2.COLOR_BGR2GRAY используется для перевода изображения в одноканальную версию
cv2.threshold()	Метод для преобразования изображения в градациях серого в двоичное изображение. cv2.THRESH_BINARY_INV – параметр задаёт порог фиксированного уровня вида двоичной инверсии. cv2.THRESH_OTSU – параметр задаёт пороговое значение по методу Оцу
cv2.getStructuringElement()	Метод задаёт структурирующий элемент kernel. MORPH_RECT – параметр обозначает прямоугольную форму ядра
cv2.morphologyEx()	Метод выполняет расширенные морфологические

	преобразования, в качестве основных операций используются эрозию и расширение. MORPH_CLOSE – используется для закрытия небольших отверстий внутри объектов переднего плана.
cv2.findContours()	Функция нахождения контуров объектов. Режим нахождения – это то, как будут сгруппированы найденные контуры. Режим нахождения: cv2.RETR_EXTERNAL – демонстрирует только крайние внешние контуры Метод упаковки: cv2.CHAIN_APPROX_NONE – полученные контуры хранятся в виде отрезков.
cv2.boundingRect()	Метод возвращает базовые координаты x и y для дальнейшей обрезки изображения.
crop()	Метод обрезает изображение по заданным координатам.
cv2.MSER_create()	Создаёт пустой объект mser (maximally stable extremal regions).
mser.detectRegions()	Метод получает максимально стабильные экстремальные области из одноканальной (серой) версии изображения.
cv2.convexHull()	Метод находит выпуклую оболочку изображения.
cv2.polylines()	Метод создаёт полигоны по вершинам, найденным предыдущей функцией.
cv2.imshow()	Метод отображает полученное изображение в отдельном окне. Размер окна автоматически подстраивается под размер изображения.
cv2.waitKey()	Команда останавливает выполнение скрипта до нажатия клавиши на клавиатуре. Параметр 0 обозначает, что метод бесконечно ожидает нажатие любой клавиши.
cv2.drawContours()	Функция отрисовки найденных контуров поверх исходного изображения.
cv2.bitwise_and(image,image, mask=mask)	Метод производит поразрядное соединение двух массивов.

Таблица использованных функций Numpy и PyTesseract

np.zeros()	Метод возвращает массив, заполненный нулями. Параметр dtype=np.uint8 обозначает тип данных, которыми заполняется массив. В данном случае это целые числа в диапазоне от 0 по 255 (размером 1 байт).
image_to_string()	Метод библиотеки Tesseract переводит изображение в текст.

Листинг программ

1:

```
import cv2
import numpy as np
import re
import pytesseract
from pytesseract import image_to_string
from PIL import Image

image_obj = Image.open("screenshot.jpg")
rgb = cv2.imread('screenshot.jpg')
```

```
    small = cv2.cvtColor(rgb, cv2.COLOR_BGR2GRAY)
    _, bw = cv2.threshold(small, 0.0, 255.0, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)

    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (20, 1))
    connected = cv2.morphologyEx(bw, cv2.MORPH_CLOSE, kernel)

    contours,          hierarchy,=cv2.findContours(connected.copy(),cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
    counter=0
    array_of_texts=[]
    for idx in range(len(contours)):
        x, y, w, h = cv2.boundingRect(contours[idx])
        cropped_image = image_obj.crop((x-10, y, x+w+10, y+h ))
        str_store = re.sub(r'([^\s\w]|_)+', '', image_to_string(cropped_image))
        array_of_texts.append(str_store)
        counter+=1
    new_array = [elem for elem in array_of_texts if elem != '\x0c']
    print(new_array)

    2:
    import cv2
    import numpy as np

    mser = cv2.MSER_create()
    img = cv2.imread('text-img.png')
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    vis = img.copy()
    regions, _ = mser.detectRegions(gray)
    hulls = [cv2.convexHull(p.reshape(-1, 1, 2)) for p in regions]
    cv2.polylines(vis, hulls, 1, (0, 255, 0))
    cv2.imshow('img', vis)
    cv2.waitKey(0)

    mask = np.zeros((img.shape[0], img.shape[1], 1), dtype=np.uint8)
    for contour in hulls:
        cv2.drawContours(mask, [contour], -1, (255, 255, 255), -1)
    text_only = cv2.bitwise_and(img, img, mask=mask)
    cv2.imshow("text only", text_only)
    cv2.waitKey(0)
```

Пример работы

4.18. Обратное преобразование перспективы

Преобразование перспективы в «вид сверху», или обратное преобразование перспективы (Inverse Perspective Mapping, сокращённо IPM) даёт возможность алгоритмам подсчитать геометрические параметры объектов на изображении без искажений, т.к. в обратной перспективе объекты не деформируются в зависимости от их положения относительно камеры.

Схема обратного преобразования перспективы представлена на рисунке. Область поиска на изображении выделяется в виде трапеции с учётом того, что при преобразовании перспективы трапеция преобразуется в прямоугольник.

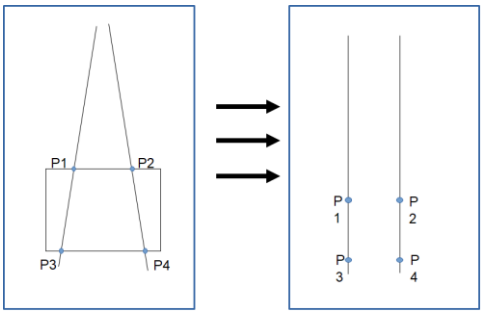
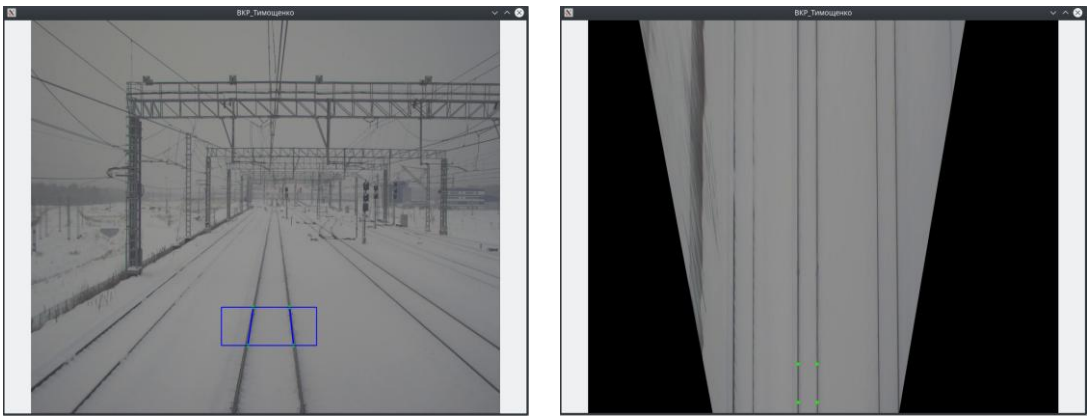


Схема преобразования перспективы



Преобразование перспективы изображения рельсовой колеи [40].

Основные выражения, реализующие преобразование:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ROI_{left} + \frac{ROI_{width} * x_{ipm}}{IPM_{width}} \\ ROI_{right} + \frac{ROI_{length} * y_{ipm}}{IPM_{height}} \end{pmatrix} (2.1)$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = T_{ext} + R_{ext} * \begin{pmatrix} x \\ y \\ z \end{pmatrix} (2.2)$$

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{x' * f_x}{z'} + c_x \\ \frac{y' * f_y}{z'} + c_y \end{pmatrix} (2.3)$$

Здесь ROI_{left} и ROI_{right} – самая левая и самая правая точка области поиска, ROI_{width} и ROI_{length} - ширина и высота области соответственно. Эти величины позволяют полностью задать область в физическом пространстве. Для вычислений обратной перспективы так же нужны параметры камеры: T_{ext}, R_{ext} – перемещение и поворот камеры, f_x, f_y – фокусные расстояния камеры, c_x и c_y – координаты точки центра проекции. Зная эти величины, каждой точке изображения (u, v) можно сопоставить точку изображения в обратной перспективе (x_{ipm}, y_{ipm}), следуя формулам 2.1,2.2 и 2.3.

Приведем пример простой программы. Программа захватывает изображение с видеокамеры. Пользователь выделяет на выходном изображении границы области, которую необходимо преобразовать в «вид сверху». После выделения отображается второе окно, отображаемое обратную перспективу выделенной области.

Используемые функции

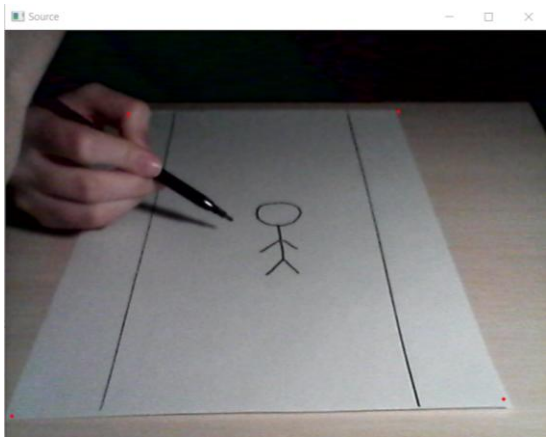
cv2.namedWindow	Создает именованное окно
cv2.VideoCapture	Захватывает изображение с видеокамеры
cv2.setMouseCallback	Функция, которая вызывает передаваемую нами функцию в ответ на любое действие мышью. Чтобы контроллировать, какое именно действие было выполненно можно использовать cv2.EVENT_LBUTTONDOWN или другие
cv2.circle	Функция позволяет рисовать окружность на передаваемом изображении. С помощью входных параметров можно регулировать позицию, цвет, размер и толщину окружности (можно также полностью заполнить окружность желаемым цветом, передав вместо толщины -1)
cv2.getPerspectiveTransform	Функция, формирующая матрицу преобразования

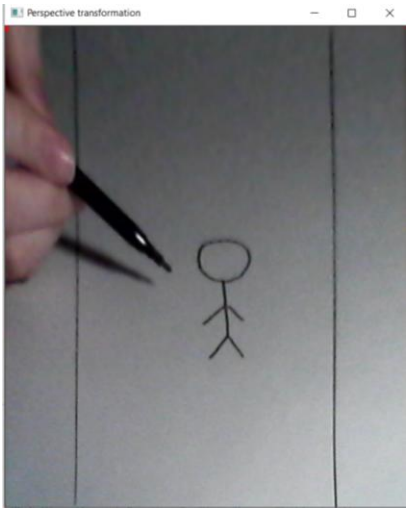
	перспективы
cv2.warpPerspective	Функция, применяющая полученную трансформацию к изображению
cv2.imshow	Используется для отображения изображения в окне
cv2.waitKey	Функция, ожидающая нажатия какой-либо кнопки
cv2.destroyAllWindows	Используется для закрытия всех созданных окон

Листинг программы

```
import cv2
import numpy as np

x_dots = []
y_dots = []

cv2.namedWindow("Source")
cap = cv2.VideoCapture(0)
_, frame = cap.read()

def draw_dot(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        x_dots.append(x)
        y_dots.append(y)

while True:
    _, frame = cap.read()
    cv2.setMouseCallback ("Source", draw_dot)

    for i in range(len(x_dots)):
        cv2.circle(frame, (x_dots[i], y_dots[i]), 2, (0, 0, 255), -1)

    if len(x_dots) == 4:
        #размер исходного изображение
        src = np.float32([[x_dots[0], y_dots[0]], [x_dots[1], y_dots[1]], [x_dots[
2], y_dots[2]], [x_dots[3], y_dots[3]]])
        #размер нового изображения
        dst = np.float32([[0, 0], [500, 0], [0, 600], [500, 600]])
        #формируем перспективу
        matrix = cv2.getPerspectiveTransform(src, dst)
        result = cv2.warpPerspective(frame, matrix, (500, 600))
        cv2.imshow("Perspective transformation", result)

    cv2.imshow("Source", frame)
    key = cv2.waitKey(1)
    if key == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

Пример работы программы





4.19 Проекции и преобразование Радона

Преобразование Радона — интегральное преобразование функции многих переменных, родственное преобразованию Фурье. Впервые введено в работе австрийского математика Иоганна Радона 1917-го года. Важнейшее свойство преобразования Радона — обратимость, то есть возможность восстанавливать исходную функцию по её преобразованию Радона.

Преобразование Радона позволяет осуществить переход из декартовой системы распределения яркостей анализируемого изображения $I(x, y)$ в полярную плоскость углового распределения яркостей на основе использования выражения:

$$R(U,\theta) = \iint I(x,y)e^{-j2\pi(x\cos(\theta)+y\sin(\theta))U}\,dxdy;\ 0 \le \theta \le \pi\ ,$$

где θ – угловая ось; U – ось проекций изображения.

Логический смысл использования подобного преобразования для анализа изображений текстур заключается в том обстоятельстве, что если у нас имеется портрет изображения текстуры $R(U, \theta)$ в полярной системе координат U и θ, то в этом портрете должна быть заложена информация о текстуре при любом угле ее ориентации, что и открывает

принципиальную возможность получения характеристик, инвариантных к влиянию поворота

Алгоритм работы программы поворота изображения с использованием преобразования Радона:

1. Считываем изображение
2. Преобразовываем его цвета в оттенки серого
3. Если разрешение высокое, изменяем размер изображения, чтобы сократить время обработки (w > 640)
4. Делаем яркость выше и ниже нуля
5. Выполняем преобразование Радона
6. Находим среднеквадратичное значение каждой строки и находим наиболее загруженную ротацию, где трансформация идеально сочетается с чередующимися темными текст и белые линии
7. Поворот
8. Сохранение с исходным разрешением

Используемые функции

Функция	Описание
imread	считывание изображения
cvtColor	используется для преобразования изображения из одного цветового пространства в другое
resize	изменение размера изображения
getRotationMatrix2D	возвращает матрицу преобразования
warpAffine	применяет аффинное преобразование к изображению
imwrite	запись изображения

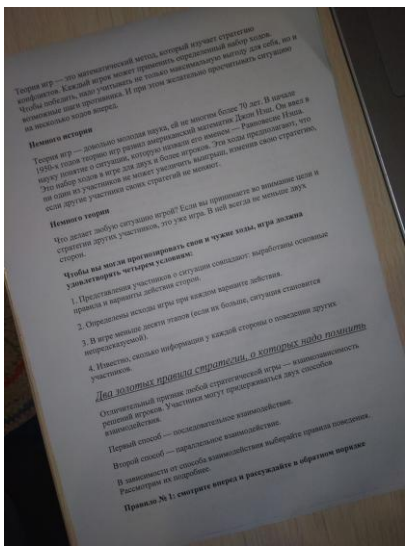
Листинг программы

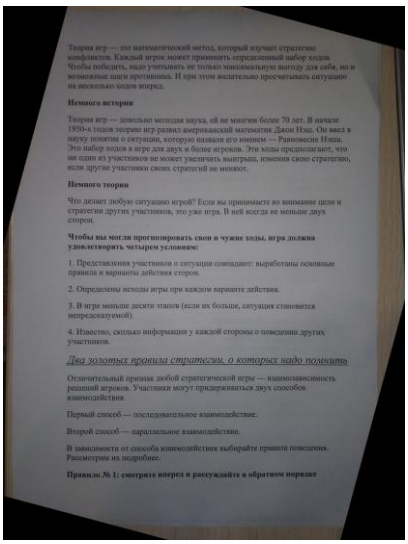
```
import numpy as np
import cv2

from skimage.transform import radon

# Вместо file_name имя к реальному файлу
```

```
filename = 'file_name'
# Загрузить файл с преобразованием в оттенки серого
img = cv2.imread(filename)
I = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
h, w = I.shape

# Если разрешение высокое, измените размер изображения, чтобы сократить время обработки.
if w > 640:
    I = cv2.resize(I, (640, int((h / w) * 640)))
I = I - np.mean(I)  # Demean; сделать яркость выше и ниже нуля
# Сделайте преобразование Радона
sinogram = radon(I)

# Нахождение среднеквадратичное значение каждой строки и найдите "наиболее загруженную" ротацию,
# где трансформация идеально сочетается с чередованием темноты
# текст и белые линии
r = np.array([np.sqrt(np.mean(np.abs(line) ** 2)) for line in sinogram.transpose()])
rotation = np.argmax(r)
print('Rotation: {:.2f} degrees'.format(90 - rotation))

# Поворот и сохранение с исходным разрешением
M = cv2.getRotationMatrix2D((w / 2, h / 2), 90 - rotation, 1)
dst = cv2.warpAffine(img, M, (w, h))
cv2.imwrite('rotated.jpg', dst)
```

Результаты работы

До: После:





4.20 Поиск совпадений между изображениями с помощью алгоритма

полного перебора SIFT дескрипторов

SIFT (Scale-Invariant Feature Transform, Масштабно-инвариантная трансформация признаков) является алгоритмом выявления признаков в компьютерном зрении. Сначала в SIFT извлекаются ключевые точки объектов из набора контрольных изображений и запоминаются в базе данных. Объект распознаётся в новом изображении путём сравнивания каждого признака из нового изображения с признаками из базы данных и нахождения признаков-кандидатов на основе евклидова расстояния между векторами признаков. Из полного набора соответствий в новом изображении отбираются поднаборы ключевых точек, которые наиболее хорошо согласуются с объектом по его местоположению, масштабу и ориентации.

Последовательный алгоритм программы:

1. Сначала загружаем два изображения: первое изображение будем искать на втором изображении.
2. Далее создаём SIFT детектор и ищем ключевые точки и дескрипторы.
3. Создаём объект BFMatcher – сопоставитель дескрипторов (массивов чисел, описывающих особенности изображения), работающий по алгоритму грубой силы(прямой перебор), с помощью него по дескрипторам сравниваем изображения.
4. Он берет один дескриптор первого изображения и сопоставляет его со всеми дескрипторами второго изображения, а затем он переходит ко второму дескриптору первого изображения и сопоставляет со всеми дескрипторами второго изображения и так далее.
5. Ключевые точки SIFT между двумя изображениями сопоставляются путем определения их ближайших соседей. Но в некоторых случаях из-за шума, второе ближайшее совпадение может показаться ближе к

первому. В этом случае мы вычисляем отношение ближайшего расстояния ко второму ближайшему расстоянию и проверяем, не превышает ли оно 0.75. Если соотношение больше 0.75, не рассматривание данное совпадение.

6. Благодаря этой проверке устраняется около 90% ложных совпадений(но в то же время 5% верных тоже могут быть устранены).
7. Выводим два изображения вместе с найденными совпадениями.

Таблица использованных функций

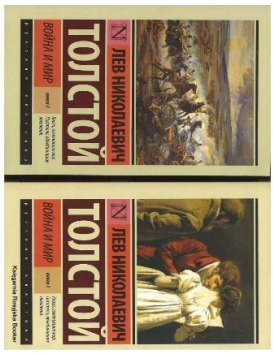
Название функции	Описание
imread()	Считывание изображения из файла
SIFT_create()	Инициализация SIFT детектора
detectAndCompute()	Нахождение ключевых точек и расчёт дескрипторов
BFMatcher()	Инициализация объекта сопоставителя дескрипторов
knnMatch()	Находит k лучших совпадений для каждого дескриптора из набора
len()	Нахождение длины массива
append()	Добавление объекта в лист
drawMatchesKnn()	Отрисовка совпадений в изображениях
imshow()	Вывод изображения

Листинг программы

```
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread('war.jpg', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('war2.jpg', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
bf = cv.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)
num_matches = len(matches)
print(f'Total number of features matches found: {num_matches}')
good = []
for m, n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])
num_good = len(good)
print(f'Total number of good matches after ratio test: {num_good}')
img3 = cv.drawMatchesKnn(img1, kp1, img2, kp2, good, None,
flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
plt.imshow(img3), plt.xticks([]), plt.yticks([]), plt.show()
```

Результаты работы программы

Исходные изображения



Результат

```
Total number of features matches found: 1398
Total number of good matches after ratio test: 682
```

Глава 5. Примеры использования алгоритмов обработки видео

5.1. Игра. Отслеживание объекта. Метод Гауссовой смеси

Разработана небольшая игра, по совмещению фигур с помощью движения. Применяется отслеживание движения с помощью метода Гауссовой смеси и обработки между кадрами видео. Дополнительно были добавлены объекты – фигуры, которые реагируют на движение и могут перемещаться.

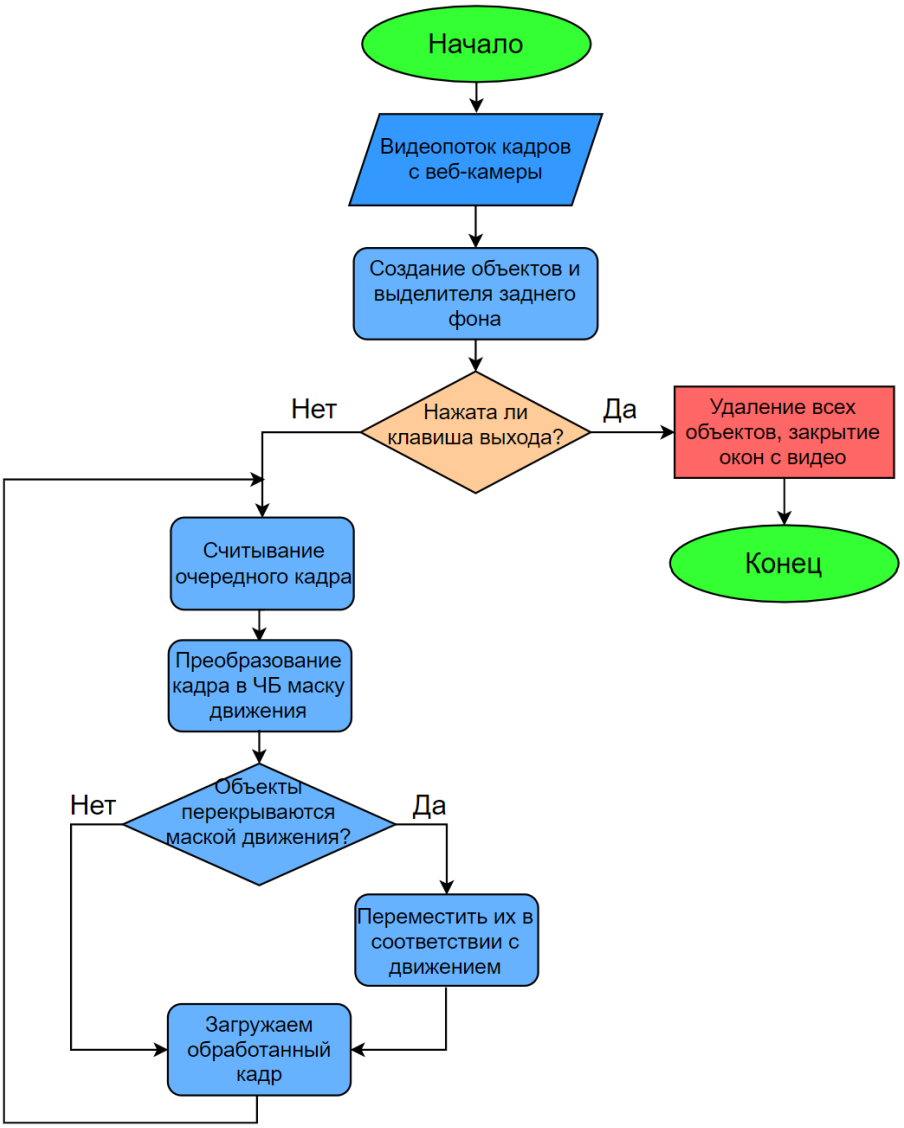
Гауссовы смеси распределений (Mixture of Gaussians, MOG) – это одна из форм оценки плотности распределения, которая является стандартным подходом к построению модели фона, использующимся для многих прикладных задач. Для каждого пикселя кадра строится функция плотности вероятности и MOG использует этот подход, чтобы различать фон и движущиеся объекты.

Таблица использованных функций

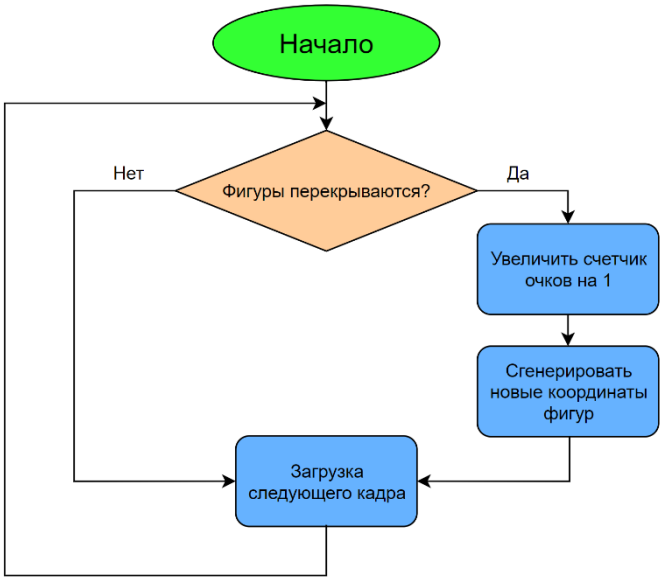
Название функции	Описание
createBackgroundSubtractorMOG2	Создание выделителя заднего фона на основе метода Гауссовой смеси (Gaussian Mixture-based Background Subtractor)
BackgroundSubtractorMOG2::apply	Создание маски движения с помощью выделителя (subtractor'a)
cvtColor	Изменение цвета изображения
threshold	Применение порогового значения к многоканальному массиву. Применяется для получения двоичного изображения в градациях серого
resize	Изменение размера изображения
putText	Вставка текста на изображение
imshow	Создание окна с изображением

Алгоритм программы

1. Алгоритм обработки движения



2. Алгоритм начисления очков



```
import cv2
import numpy as np
import math
import random

# Класс Объекта квадрата для движения
class Square:

    # Конструктор
    def __init__(self, start_picture, start_x, start_y, size):
        self.start_picture = start_picture
        self.size = size
        self.picture = cv2.resize(self.start_picture, (size, size))
        img2gray = cv2.cvtColor(self.picture, cv2.COLOR_BGR2GRAY)
        _, picture_mask = cv2.threshold(img2gray, 1, 255, cv2.THRESH_BINARY)
        self.picture_mask = picture_mask
        self.x = start_x
        self.y = start_y
        self.on_mask = False

    # Геттеры
    def get_x(self):
        return self.x

    def get_y(self):
        return self.y

    # Сеттеры
    def set_x(self, value):
        self.x = value

    def set_y(self, value):
        self.y = value

    # Вставка объекта в кадр
    def insert_square(self, frame):
        coordinates = frame[self.y:self.y + self.size, self.x:self.x + self.size]
        coordinates[np.where(self.picture_mask)] = 0
        coordinates += self.picture

    # Обновление позиции на экране
    def update_position(self, mask):
        height, width = mask.shape

        # Проверка наложения объекта на движущуюся часть
        coordinates = mask[self.y:self.y + self.size, self.x:self.x + self.size]
        overlapping = np.any(coordinates[np.where(self.picture_mask)])
```

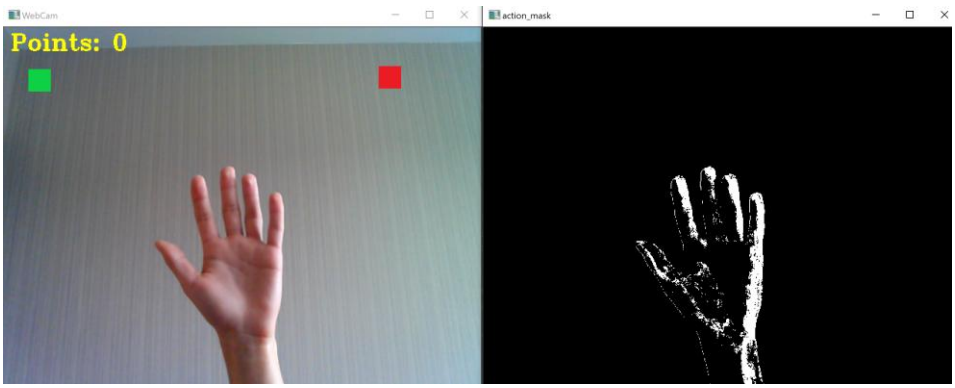
```
        # Если на объект накладывается движение, то выбираем направление
        if overlapping:

            # Подготовка к движению
            best_delta_x = 0
            best_delta_y = 0
            best_fit = np.inf

            # Для всех 8 направлений
            for _ in range(8):

                # Формирование небольшого рандомного смещения
                delta_x = np.random.randint(-15, 15)
                delta_y = np.random.randint(-15, 15)

                # Проверка, что при смещении объект не вылетит за кадр
                if self.y + self.size + delta_y > height or \
                    self.y + delta_y < 0 or \
                    self.x + self.size + delta_x > width or \
                    self.x + delta_x < 0:
                    continue

                # Подсчет того, на сколько большое перекрытие с движением
                coordinates = mask[self.y + delta_y:self.y + delta_y +
                 self.size, self.x + delta_x:self.x + delta_x + self.size]
                overlapping = np.count_nonzero(
                    coordinates[np.where(self.picture_mask)])

                # Если нет движения, стоим на месте
                if overlapping == 0:
                    self.x += delta_x
                    self.y += delta_y
                    return
                # Если же движение есть, смещаемся в нужном направлении
                elif overlapping < best_fit:
                    best_fit = overlapping
                    best_delta_x = delta_x
                    best_delta_y = delta_y

            # После подсчета всех направлений, выполняем итоговое смещение
            if best_fit < np.inf:
                self.x += best_delta_x
                self.y += best_delta_y
                return

# Получаем изображение с веб-камеры
camera = cv2.VideoCapture(0)

# Устанавливаем ширину и высоту кадра 640X480
```

```
camera.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

# Создаем выделитель заднего фона
background_subtractor = cv2.createBackgroundSubtractorMOG2()

# Создаем объекты квадратов по изображениям
green_picture = cv2.imread('square_green.png')
red_picture = cv2.imread('square_red.png')
square_green = Square(green_picture, 40, 50, 30)
square_red = Square(red_picture, 600, 50, 30)
# Очки
points = 0

# Бесконечный цикл обработки кадров, пока не завершим выходом
while True:

    # Получение очередного кадра с веб-камеры
    _, frame = camera.read()

    # Поворот кадра
    frame = cv2.flip(frame, 1)

    # Получаем маску движения с помощью выделителя
    action_mask = background_subtractor.apply(frame)

    # Раскрашиваем маску в черно-белое по движению
    _, action_mask = cv2.threshold(action_mask, 250, 255, cv2.THRESH_BINARY)

    # Обновляем положение объектов по маске движения
    square_green.update_position(action_mask)
    square_red.update_position(action_mask)

    # Вставляем объекты в кадр
    square_green.insert_square(frame)
    square_red.insert_square(frame)

    # Обновляем текст с кол-вом очков
    text = f"Points: {points}"
    cv2.putText(frame, text, (10, 30),
     cv2.FONT_HERSHEY_TRIPLEX, 1, (0, 255, 255), 2)

    # Показывем кадр в окне веб-камеры
    cv2.imshow('WebCam', frame)
    # Показываем кадр маски в другом окне
    cv2.imshow('action_mask', action_mask)

    # Если нажата клавиша q, то выход
    if cv2.waitKey(1) == ord('q'):
```

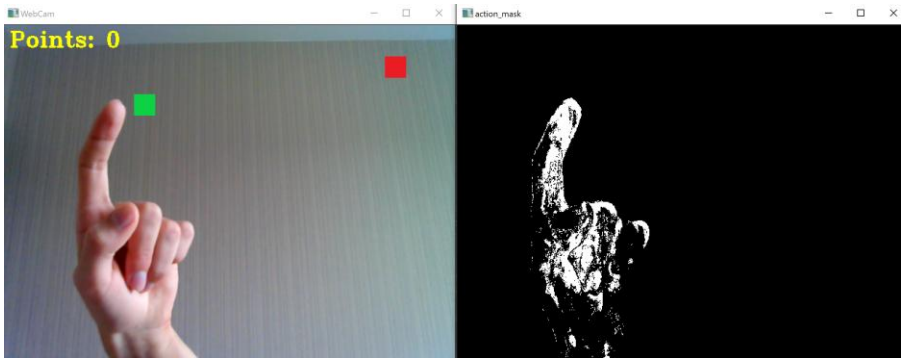
```
            break

        # Подсчет расстояния между объектами, определение столкновения
        if math.sqrt(pow(square_green.get_x() - square_red.get_x(), 2)
         + pow(square_green.get_y() - square_red.get_y(), 2)) < 30:
            # Если столкнулись, добавляем очки
            points += 1
            # Генерируем новые местоположения
            one_x = random.randint(30, 600)
            one_y = random.randint(30, 450)
            two_x = random.randint(30, 600)
            two_y = random.randint(30, 450)

            # Выводим сгенерированные координаты
            print("GREEN: ", one_x, one_y, " RED: ", two_x, two_y)

            # Устанавливаем новые местоположения
            square_green.set_x(one_x)
            square_green.set_y(one_y)
            square_red.set_x(two_x)
            square_red.set_y(two_y)


# Завершаем работу и удаляем все окна
camera.release()
cv2.destroyAllWindows()
```

Скриншоты

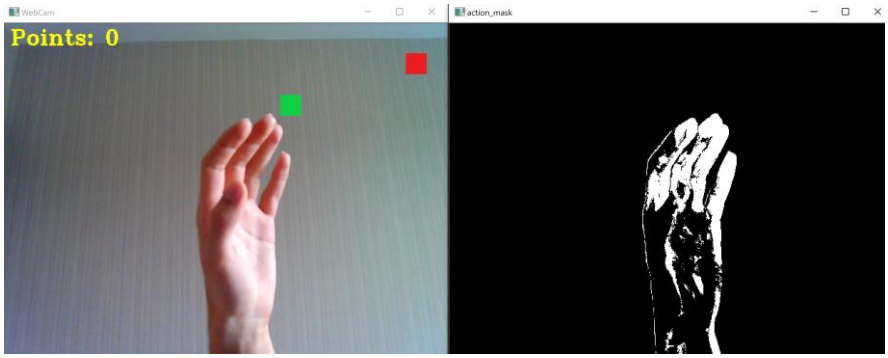
1. Старт



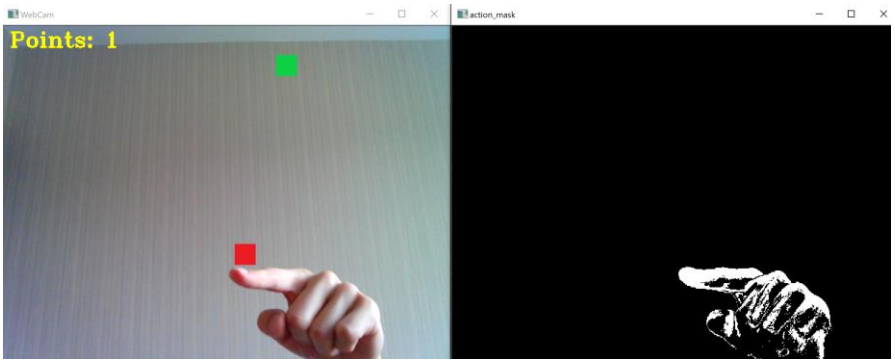
2. Перемещение объекта



3. Совмещение объектов

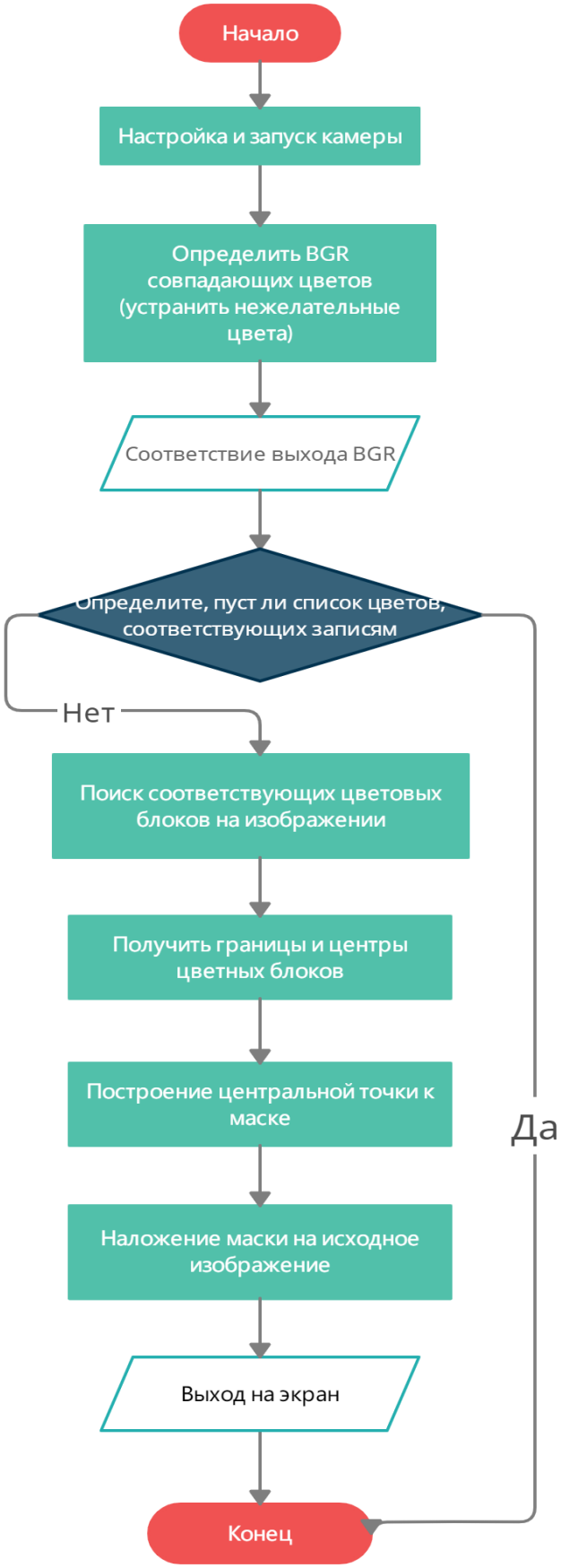


4. Объекты совмещены, добавлено одно очко, у объектов новые координаты



5.2. Рисование путем перемещения объекта перед видеокамерой

Рисование в OpenCV достигается с помощью функций, предоставляемых OpenCV, таких как определение контура, согласование цветов и графика рисования. Ниже представлена программа, которая позволяет рисовать в рабочем окне путем перемещения объекта перед видеокамерой.


Начало
Настройка и запуск камеры
Определить BGR совпадающих цветов (устранить нежелательные цвета)
Соответствие выхода BGR
Определите, пуст ли список цветов, соответствующих записям
Нет
Да
Поиск соответствующих цветовых блоков на изображении
Получить границы и центры цветных блоков
Построение центральной точки к маске
Наложение маски на исходное изображение
Выход на экран
Конец

numpy	
numpy.zeros((CAM_HEIGHT, CAM_WIDTH, 3), dtype=np.uint8)	Generate a black image of numpy.uint8 type with the length and width as the parameters of the camera
numpy.array([h_min, s_min, v_min])	Create an array based on the values of the parameters

cv2	
cv2.VideoCapture(CAM_ID)	Opens a video file or a capturing device or an IP video stream for video capturing with API Preference
cam.set(3, CAM_WIDTH)	Switches exceptions mode (Set camera width)
cam.set(4, CAM_HEIGHT)	Switches exceptions mode (Set camera height)
cam.set(10, CAM_BRIGHTNESS)	Switches exceptions mode (Set camera brightness)
cam.read()	Grabs, decodes and returns the next video frame
cv2.namedWindow("HSY-TrackBars")	Creates a window. The function namedWindow creates a window that can be used as a placeholder for images and trackbars. Created windows are referred to by their names
cv2.resizeWindow("HSY-TrackBars", CAM_WIDTH, CAM_HEIGHT)	Resizes the window to the specified size
cv2.createTrackbar("Hue Min", "HSY-TrackBars", 0, 179, empty)	Creates a trackbar and attaches it to the specified window. The function createTrackbar creates a trackbar (a slider or range control) with the specified name and range, assigns a variable value to be a position synchronized with the trackbar and specifies the callback function onChange to be called on the trackbar position change. The created trackbar is displayed in the specified window winname
cv2.getTrackbarPos("Hue Min", "HSY-TrackBars")	Returns the trackbar position
cv2.cv2tColor(img, cv2.COLOR_BGR2HSV)	Converting the color space of an image to HSV
cv2.inRange(img_HSV, lower, upper)	Checks if array elements lie between the elements of two other arrays
cv2.bitwise_and(img, img, mask=mask)	computes bitwise conjunction of the two arrays (dst = src1 & src2) Calculates the per-element bit-wise conjunction of two arrays or an array and a scalar
cv2.circle(gl_img_result, (x, y), PRINT_SIZE, PRINT_COLOR, -1)	Draws a circle
cv2.drawContours(gl_img_result, event, -1, PRINT_COLOR, 3)	Draws contours outlines or filled contours.
cv2.arcLength(event, True)	Calculates a contour perimeter or a curve length
cv2.approxPolyDP(event, 0.02 * peri_img, True)	Approximates a polygonal curve(s) with the specified precision

cv2.boundingRect(approx)	Calculates the up-right bounding rectangle of a point set or non-zero pixels of gray-scale image
cv2.addWeighted(img_row, 1, gl_path_mask, 1, 0)	Calculates the weighted sum of two arrays
cv2.imshow("Row camera", img)	Displays an image in the specified window
cv2.waitKey(1)	Waits for a pressed key

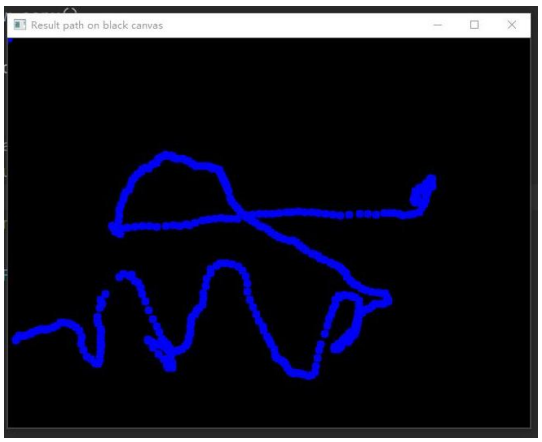
Листинг программы

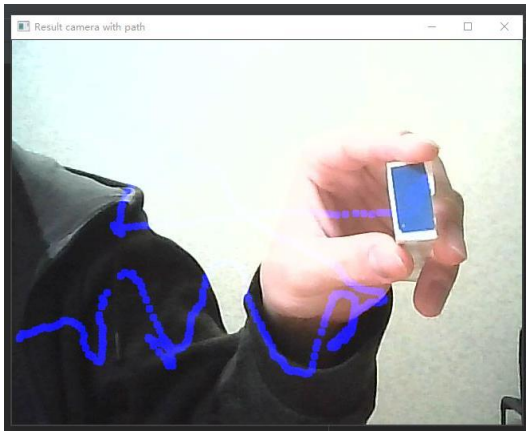
```
# -*- coding: utf-8 -*-
# @Time    : 2021/3/6
# @Author  : Мэн Цзянин
# @FileName: KP15.py
# @Software: Pycharm
# @Versions: v0.3
# @Github   : https://github.com/NekoSilverFox
# ~~~~~~~~~~~~~~~~~~~

import cv2 as cv
import numpy as np
from stackImages import stackImages

# Определяет некоторые из необходимых констант
CAM_WIDTH = 640
CAM_HEIGHT = 480
CAM_BRIGHTNESS = 150
CAM_ID = 0
PRINT_SIZE = 5
PRINT_COLOR = (255, 0, 0)
gl_path_mask = np.zeros((CAM_HEIGHT, CAM_WIDTH, 3), dtype=np.uint8)

def empty(none):
    pass

# Для цветного экранирования
def color_picker():
    # Настройка размера камеры
    cam = cv.VideoCapture(CAM_ID)
    cam.set(3, CAM_WIDTH)
    cam.set(4, CAM_HEIGHT)
    cam.set(10, CAM_BRIGHTNESS)

    # Создание трекер-бара
    cv.namedWindow("HSY-TrackBars")
    cv.resizeWindow("HSY-TrackBars", CAM_WIDTH, CAM_HEIGHT)

    cv.createTrackbar("Hue Min", "HSY-TrackBars", 0, 179, empty)
    cv.createTrackbar("Sat Min", "HSY-TrackBars", 0, 255, empty)
    cv.createTrackbar("Val Min", "HSY-TrackBars", 0, 255, empty)

    cv.createTrackbar("Hue Max", "HSY-TrackBars", 179, 179, empty)
    cv.createTrackbar("Sat Max", "HSY-TrackBars", 255, 255, empty)
    cv.createTrackbar("Val Max", "HSY-TrackBars", 255, 255, empty)

    while True:
        is_success, img = cam.read()
        img_HSV = cv.cvtColor(img, cv.COLOR_BGR2HSV)
```

```
        h_min = cv.getTrackbarPos("Hue Min", "HSY-TrackBars")
        h_max = cv.getTrackbarPos("Hue Max", "HSY-TrackBars")

        s_min = cv.getTrackbarPos("Sat Min", "HSY-TrackBars")
        s_max = cv.getTrackbarPos("Sat Max", "HSY-TrackBars")

        v_min = cv.getTrackbarPos("Val Min", "HSY-TrackBars")
        v_max = cv.getTrackbarPos("Val Max", "HSY-TrackBars")
        print(h_min, h_max, s_min, s_max, v_min, v_max)

        # Установите нежелательные цвета на черный, а желаемые - на белый
        lower = np.array([h_min, s_min, v_min])
        upper = np.array([h_max, s_max, v_max])
        mask = cv.inRange(img_HSV, lower, upper)
        img_result = cv.bitwise_and(img, img, mask=mask)
        mask = cv.cvtColor(mask, cv.COLOR_GRAY2BGR)

        img_stack = stackImages(0.6, ([img, mask, img_result]))
        cv.imshow("Stack images", img_stack)
        cv.waitKey(1)

# Путь для рисования цветных блоков
def print_path(x, y):
    cv.circle(gl_img_result, (x, y), PRINT_SIZE, PRINT_COLOR, -1)
    cv.circle(gl_path_mask, (x, y), PRINT_SIZE, PRINT_COLOR, -1)

# Используется для поиска цветового блока заданного цвета.
def find_color(img_input, prick_colors_list):
    img_HSV = cv.cvtColor(img_input, cv.COLOR_BGR2HSV)
    index_color = 0

    for color in prick_colors_list:
        lower = np.array(color[0:3])
        upper = np.array(color[3:6])
        mask = cv.inRange(img_HSV, lower, upper)

        x, y = get_contours(mask)
        print_path(x, y)

        index_color += 1
        if index_color > 3:
            print("[ERROR] In dead loop!")

        # print(mask)
        cv.imshow("Mask", mask)

def get_contours(img):
    contours, hierarchy = cv.findContours(img, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_N
ONE)
    x = 0
    y = 0
    w = 0
    h = 0
    for event in contours:
        area_img = cv.contourArea(event)
        if area_img > 500:
            cv.drawContours(gl_img_result, event, -1, PRINT_COLOR, 3)
            peri_img = cv.arcLength(event, True)
            approx = cv.approxPolyDP(event, 0.02 * peri_img, True)
            x, y, w, h = cv.boundingRect(approx)
```

```
            print("Print point on [%d, %d]" % (x, y))
    return (x + w // 2), (y + h // 2)

# color_picker()

# 设置摄像头的大小
# Настройка размера камеры
cam = cv.VideoCapture(CAM_ID)
cam.set(3, CAM_WIDTH)
cam.set(4, CAM_HEIGHT)
cam.set(10, CAM_BRIGHTNESS)
output = cv.VideoWriter("output.avi", cv.VideoWriter_fourcc("M", "J", "P", "G"), 2
9, (CAM_WIDTH, CAM_HEIGHT))

# Используйте список для записи цветов, которые могут быть сопоставлены
picker_colors_list = [  # [0, 103, 255, 179, 255, 255],  # Red
    [97, 34, 93, 179, 255, 255]]  # Blue

while True:
    is_success, img_row = cam.read()
    if not is_success:
        print("[ERROR] Can not read camera!")
        break

    gl_img_result = img_row.copy()

    find_color(img_row, picker_colors_list)
    img_result_with_path = cv.addWeighted(img_row, 1, gl_path_mask, 1, 0)

    # Отображение изображений в виде окна
    # cv.imshow("Row camera", img)
    cv.imshow("Catch contours", gl_img_result)
    cv.imshow("Result path on black canvas", gl_path_mask)
    cv.imshow("Result camera with path", img_result_with_path)
    output.write(img_result_with_path)

    # Выход из программы с помощью клавиши `q` на клавиатуре
    if cv.waitKey(1) & 0xFF == ord('q'):
        break
cam.release()
output.release()
```

Скриншоты





5.3. Сканер документов

Сканер документов, используя встроенную камеру, может распознавать и находить листы бумаги (документы). На найденных листах он меняет их перспективу и изменяет цвет на более читабельный, в некоторых случаях при хорошем освещении и качестве камеры может заменить обычных сканер принтера, но при этом он еще удобно и быстро сканирует, и показывает результат в реальном времени на экране. Алгоритм работы:

1. Считываем изображение с видео камеры устройства
2. Изменяем цветовое пространство для лучшего использования
3. Применим Гауссовую фильтрацию изображения из библиотеки Open CV
4. Обнаруживаем края объектов на изображении
5. Делаем размытие краем и их увеличение для более простого отличия границы
6. Рисуем контура на предварительных изображениях
7. Аппроксимируем все контура к более простым фигурам.
8. Находим такой контур, который удовлетворяет условию: - имеет наибольшую площадь на всем изображении (фильтруем те контура, которые имеют маленькую площадь (для ускорения дальнейших проверок)); - имеет 4 угла.
9. На основе этих данных изменяем положение фигуры, выпрямляем, и ставим в свою перспективу, обрезаем все лишнее.

Для изменения перспективы использовалась функции cv2.getPerspectiveTransfor и cv2.warpPerspective, а для изменения изображения на более читабельное и удобное для просмотра cv2.adaptiveThreshold.

1. cv2.getPerspectiveTransfor(src, dst) – подсчет матрицы преобразования перспективы

   src – начальные крайние точки объекта.

   dst – то во что хотим преобразовать.

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2, 3$$

2. cv2.warpPerspective(src, M, dsize) – в ходе выполнения вернется преобразованная фотография.

   scr – передаем изображение.

   M – матрица трансформации 3*3.

   dsize – размер конечного изображения.

$$\texttt{dst}(x, y) = \texttt{src}\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right)$$

3. cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C) – пороговое преобразование (в данной программе применяется в конечном выводе изображение, более читабельный и яркий контраст)

   src – передаем изображение.

   maxValue – максимальное значение пикселей.

   adaptiveMethod – адаптивный алгоритм порогового значения.

   thresholdType – тип порогового значения.

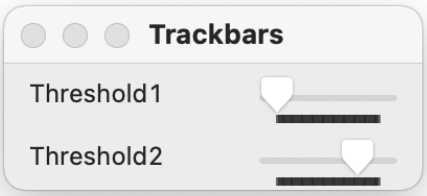
   blockSize – Размер пиксельной окрестности, используемой для вычисления порогового значения.

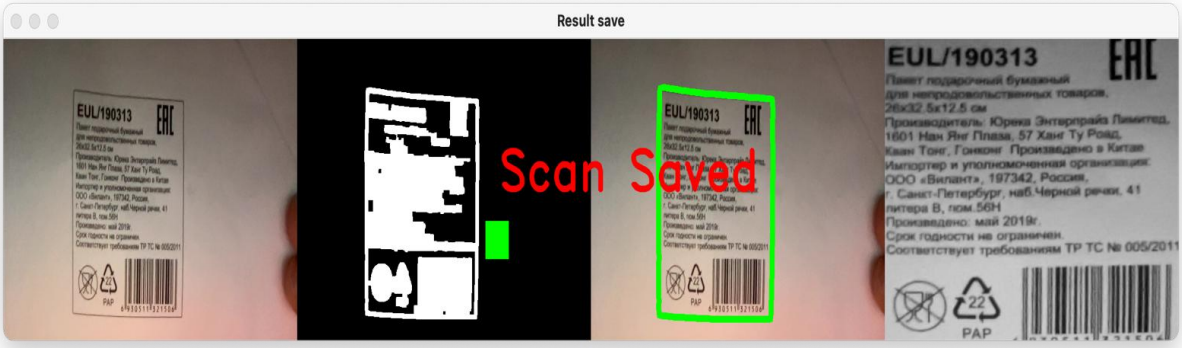
   C – Константа, вычитаемая из среднего или взвешенного среднего.

Используемые функции

cap = cv2.VideoCapture(0), cap.read()	Захватывает изображение с видеокамеры, чтение кадра.
cv2.cvtColor	Изменение цветового пространства изображения
cv2.GaussianBlur	Функция позволяет сделать фильтрацию используя гауссово ядро
cv2.Canny	Функция обнаружения краев. По шагам: 1. Ищем градиент интенсивности изображения 2. Не максимальное подавление (убирание ненужных пикселей путем проверки на локальный максимум в своей окрестности) 3. Порог гистерезиса
cv2.dilate	Функция расширение границы
cv2.erode	Функция размытия
cv2.findContours	Функция нахождение контуров объектов (нужна предварительная подготовка изображение, функции выше)
cv2.drawContours	Функция рисование контуров на фотографии
cv2.contourArea	Функция контурной области
cv2.arcLength	Функция контурный периметр (длина дуги) нужна для нахождения листа бумаги (документа)
cv2.approxPolyDP	Аппроксимация фигуры к более простой менее ломанной
cv2.imshow	Используется для отображения изображения в окне
cv2.destroyAllWindows	Используется для закрытия всех созданных окон

Пример выполнения программы





```
import cv2
import numpy as np
import utlis

cap = cv2.VideoCapture(0)
cap.set(10, 160)
heightImg = 480
widthImg = 640

utlis.initializeTrackbars()
count = 0

while True:
    success, img = cap.read()
    img = cv2.resize(img, (widthImg, heightImg))
    imgBlank = np.zeros((heightImg, widthImg, 3), np.uint8)
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray, (5, 5), 1)
    thres = utlis.valTrackbars()
    imgThreshold = cv2.Canny(imgBlur, thres[0], thres[1])
    kernel = np.ones((5, 5))
    imgDial = cv2.dilate(imgThreshold, kernel, iterations=2)
    imgThreshold = cv2.erode(imgDial, kernel, iterations=1)

    imgContours = img.copy()
    imgBigContour = img.copy()
    contours, hierarchy = cv2.findContours(imgThreshold, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(imgContours, contours, -1, (0, 255, 0), 10)

    # FIND THE BIGGEST COUNTOUR
    biggest, maxArea = utlis.biggestContour(contours)
    if biggest.size != 0:
        biggest = utlis.reorder(biggest)
        cv2.drawContours(imgBigContour, biggest, -1, (0, 255, 0), 20)
        imgBigContour = utlis.drawRectangle(imgBigContour, biggest, 2)
        pts1 = np.float32(biggest)
        pts2 = np.float32([[0, 0], [widthImg, 0], [0, heightImg], [widthImg,
heightImg]])
        matrix = cv2.getPerspectiveTransform(pts1, pts2)
        imgWarpColored = cv2.warpPerspective(img, matrix, (widthImg, heightImg))

        imgWarpColored = imgWarpColored[20:imgWarpColored.shape[0] - 20,
20:imgWarpColored.shape[1] - 20]
        imgWarpColored = cv2.resize(imgWarpColored, (widthImg, heightImg))

        imgWarpGray = cv2.cvtColor(imgWarpColored, cv2.COLOR_BGR2GRAY)
        imgAdaptiveThre = cv2.adaptiveThreshold(imgWarpGray, 255, 1, 1, 7, 2)
```

```
        imgAdaptiveThre = cv2.bitwise_not(imgAdaptiveThre)
        imgAdaptiveThre = cv2.medianBlur(imgAdaptiveThre, 3)

        imageArray = ([img, imgThreshold, imgContours, imgAdaptiveThre])

    else:
        imageArray = ([img, imgThreshold, imgContours, imgBlank])

    stackedImage = utlis.stackImages(imageArray, 1)
    cv2.imshow("Result", stackedImage)

    # SAVE IMAGE WHEN 's' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('s'):
        cv2.imwrite("Scanned/myImage" + str(count) + ".jpg", img)
        cv2.rectangle(stackedImage, ((int(stackedImage.shape[1] / 2) - 230),
int(stackedImage.shape[0] / 2) + 50),
                      (1100, 350), (0, 255, 0), cv2.FILLED)
        cv2.putText(stackedImage, "Scan Saved", (int(stackedImage.shape[1] / 2) - 200,
int(stackedImage.shape[0] / 2)),
                    cv2.FONT_HERSHEY_DUPLEX, 3, (0, 0, 255), 5, cv2.LINE_AA)
        cv2.imshow('Result save', stackedImage)
        cv2.waitKey(300)
        count += 1

import cv2
import numpy as np


def stackImages(imgArray, scale):
    rows = len(imgArray)
    rowsAvailable = isinstance(imgArray[0], list)
    width = imgArray[0][0].shape[1]
    height = imgArray[0][0].shape[0]
    for x in range(0, rows):
        imgArray[x] = cv2.resize(imgArray[x], (0, 0), None, scale, scale)
        if len(imgArray[x].shape) == 2:
            imgArray[x] = cv2.cvtColor(imgArray[x], cv2.COLOR_GRAY2BGR)
    if rowsAvailable:
        imageBlank = np.zeros((height, width, 3), np.uint8)
        hor = [imageBlank] * rows
        hor_con = [imageBlank] * rows
        for x in range(0, rows):
            hor[x] = np.hstack(imgArray[x])
            hor_con[x] = np.concatenate(imgArray[x])
        ver = np.vstack(hor)
    else:
        hor = np.hstack(imgArray)
        ver = hor
    return ver
```

```
def reorder(myPoints):
    myPoints = myPoints.reshape((4, 2))
    myPointsNew = np.zeros((4, 1, 2), dtype=np.int32)
    add = myPoints.sum(1)

    myPointsNew[0] = myPoints[np.argmin(add)]
    myPointsNew[3] = myPoints[np.argmax(add)]
    diff = np.diff(myPoints, axis=1)
    myPointsNew[1] = myPoints[np.argmin(diff)]
    myPointsNew[2] = myPoints[np.argmax(diff)]

    return myPointsNew


def biggestContour(contours):
    biggest = np.array([])
    max_area = 0
    for i in contours:
        area = cv2.contourArea(i)
        if area > 5000:
            peri = cv2.arcLength(i, True)
            approx = cv2.approxPolyDP(i, 0.02 * peri, True)
            if area > max_area and len(approx) == 4:
                biggest = approx
                max_area = area
    return biggest, max_area


def drawRectangle(img, biggest, thickness):
    cv2.line(img, (biggest[0][0][0], biggest[0][0][1]), (biggest[1][0][0],
biggest[1][0][1]), (0, 255, 0), thickness)
    cv2.line(img, (biggest[0][0][0], biggest[0][0][1]), (biggest[2][0][0],
biggest[2][0][1]), (0, 255, 0), thickness)
    cv2.line(img, (biggest[3][0][0], biggest[3][0][1]), (biggest[2][0][0],
biggest[2][0][1]), (0, 255, 0), thickness)
    cv2.line(img, (biggest[3][0][0], biggest[3][0][1]), (biggest[1][0][0],
biggest[1][0][1]), (0, 255, 0), thickness)

    return img


def nothing(x):
    pass

def initializeTrackbars(intialTracbarVals=0):
    cv2.namedWindow("Trackbars")
    cv2.resizeWindow("Trackbars", 360, 240)
```

```
    cv2.createTrackbar("Threshold1", "Trackbars", 0, 255, nothing)
    cv2.createTrackbar("Threshold2", "Trackbars", 200, 255, nothing)

def valTrackbars():
    Threshold1 = cv2.getTrackbarPos("Threshold1", "Trackbars")
    Threshold2 = cv2.getTrackbarPos("Threshold2", "Trackbars")
    src = Threshold1, Threshold2
    return src
```

5.4 Создание и чтение QR-кодов

QR-code (Quick Response Code или код быстрого реагирования) – это тип матричных штрих кодов, но в отличие от обычных штрихкодов, они состоят из маленьких квадратиков и способен нести в себе намного больше информации. Их особенностью также является то, что его можно считывать в двух направлениях, в то время как обычный только в один. Первая система QR кодов была придумана в 1944 году в Японии, но широкое признание получило только в 2012 году и к сегодняшнему дню эти коды используются повсюду.

Для чтения QR-кодов будет использована библиотека OpenCV, где уже встроены функции считывания кодов. В нашем случае считывание будет происходить с камеры компьютера. Также если в QR коде был зашиврована ссылка на какой-либо сайт, то происходит автоматический переход на эту страницу с помощью библиотек re (для определения ссылки с помощью регулярного выражения), qrcode(для создания QR-кодов) и webbrowser (для открытия ссылок)

Алгоритм работы:

1) Подключение всех нужных библиотек: OpenCV, webbrowser, re, qrcode
2) У пользователя на выбор 3 функции: создать QR-код, считать QR-код и выход из программы
3) Чтобы создать QR-код пользователь просто вводит что угодно, в том числе и url-адреса и эта информация кодируется и сохраняется в формате png

4) Дальше пользователь может ввести слово Read, если хочет декодировать код. После этого выбирает или считать с камеры или с фотографии. Если с камеры, то он должен прописать “Cam”, если с картинки, то просто название этой фотографии
5) Если пользователь прописал “Cam” то происходит захват камеры компьютера и вывод видео на экран
6) Считывание QR-кода с камеры или изображения и его декодирование. Если декодированная информация является url-адресом, то она открывается в браузере компьютера. Если другая информация, то она просто выводится в консоль.
7) Также был предотвращен «спам» информации, суть которого будет описана далее.
8) Выход из считывателя происходит посредством нажатия кнопки Esc.
9) Выход из программы происходит, когда пользователь прописал exit

Список использованных функций

Название функции	Описание
qrcode.make	Создание qr-кода по введенному тексту
save	Сохранение созданного QR-кода
videoCapture	Захват видео камеры компьютера
QRCodeDetector	Создание детектора QR-кодов
read	Считывание кадра с камеры. Возвращает два аргумента: 1) Bool-если True, то считывание кадра прошло успешно 2) сам кадр
detectAndDecode	Получает на вход наш кадр и декодирует QR-код. Возвращает 3 аргумента: 1) декодированная информация 2) массив вершин нашего найденного QR-кода 3) изображение, содержащее выпрямленный и бинаризованный QR-код
re.search	С помощью регулярного выражения проверяет нашу информацию на url-адрес
webbrowser.open_new	Открытие ссылки в браузере
flip	Отзеркаливаем вертикально наше видео
imshow	Показ кадра
release	Освобождение аппаратных ресурсов
destroyAllWindows	Закрытие всех окон

Листинг программы

```
     import cv2
import webbrowser
import re
import qrcode

print("Чтобы сосздать QR-код просто напишите то, что хотите закодировать.")
print("Чтобы считать QR-код напишите: \"Read\" ")
```

```
print("Чтобы выйти напишите exit")
data = ""
while data != "exit":
    data = input()
    if(data != "Read"):
        img_name = "qr.png"
        img = qrcode.make(data)
        img.save(img_name)
        print("QR-код создан")
    else:
        print("Чтобы считать QR-код с картинки напишите название этой
картинки.(Например \"qr.png\")")
        print("Если хотите считать с камеры, то напишите \"Cam\"")
        inp = input()
        if(inp == "Cam"):
            windowName = "QR code detector cam"
            cv2.namedWindow(windowName)
            cap = cv2.VideoCapture(0)
            detector = cv2.QRCodeDetector()
            oldData = ""
            count = 0
            while True:
                _, img = cap.read() # первый выходной параметр в программе не
требуется, поэтому ставим прочерк
                data, _, _ = detector.detectAndDecode(img) # второй и третий
выходной параметр также не требуется
                if data and data != oldData:
                    if re.search(r'((http|https)\:\/\/)?[a-zA-Z0-9\.\/\?\:@\-
_=#]+\.([a-zA-Z]){2,6}([a-zA-Z0-9\.\&\/\?\:@\-_=#])',data):
                        webbrowser.open_new(data)
                    else:
                        print("QR code message-->", data)
                    oldData = data
                img = cv2.flip(img, 1)
                cv2.imshow(windowName, img)
                count += 1
                if count == 100:
                    oldData = ""
                    count = 0

                if cv2.waitKey(1) & 0xFF == 27:
                    break
            cap.release()
            cv2.destroyAllWindows()
        else:
            img = cv2.imread(inp)
            detector = cv2.QRCodeDetector()
            data, _, _ = detector.detectAndDecode(img)
            if re.search(r'((http|https)\:\/\/)?[a-zA-Z0-9\.\/\?\:@\-_=#]+\.([a-zA-
```

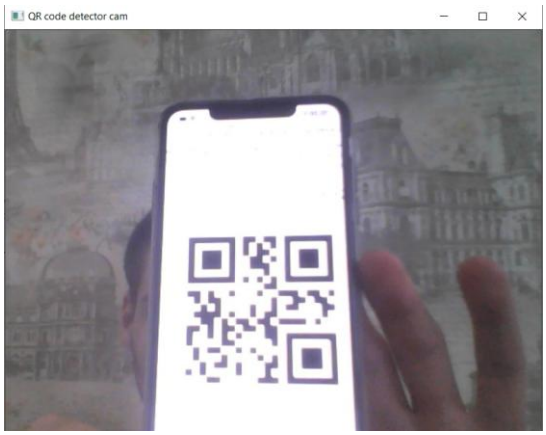
```
Z]){2,6}([a-zA-Z0-9\.\&\/\?\:@\-_=#])',data):
                webbrowser.open_new(data)
            else:
                print("QR code message-->", data)
```

Описание «защиты от спама»:

У нас сохраняется последняя прочитанная информация и при следующем считывании проверяется, не повторяются ли данные. Если QR-код был тот же самый, то повторно информация не покажется. Также было добавлено «время жизни» считанной информации, по истечении которого oldData обнуляется и код может опять считаться, даже если предыдущий раз был считан тот же самый код. При считывании подряд разных кодов никаких ограничений не накладываются.

Примеры использования:

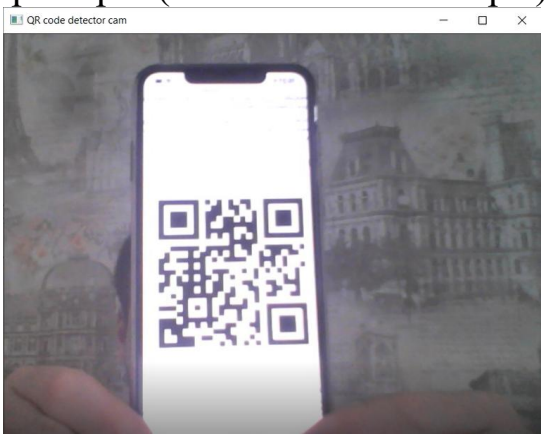
Пример 1 (считывание с камеры):



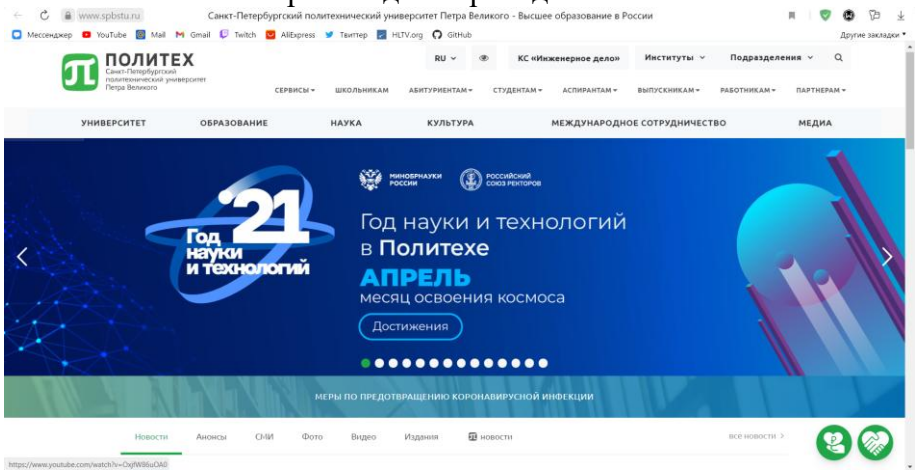
Вывод:

```
C:\Users\arsma\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/arsma/PycharmProjects/pythonProject1/main.py
QR code message--> Привет!!
```

Пример 2 (считывание с камеры):



Происходит переход на сайт



Пример 3 (создание QR-кода):

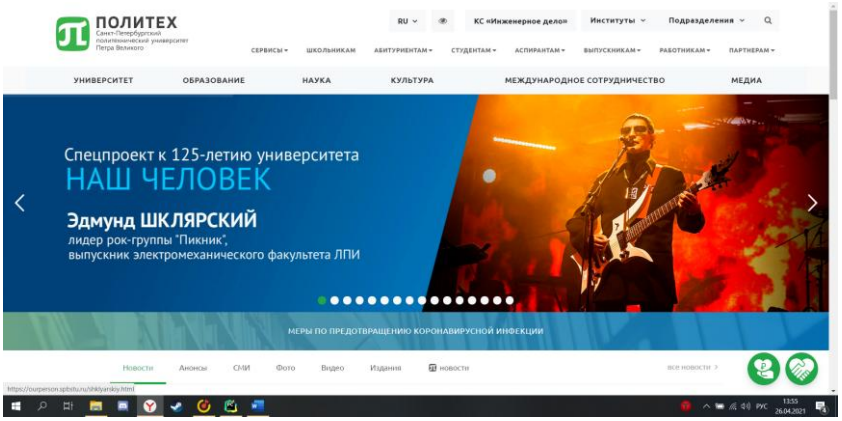
```
Чтобы сосздать QR-код просто напишите то, что хотите закодировать.
Чтобы считать QR-код напишите: "Read"
Чтобы выйти напишите exit
https://www.spbstu.ru/
QR-код создан
```

Полученный код:

Пример 4 (считывание с картинки):

```
Read
Чтобы считать QR-код с картинки напишите название этой картинки.(Например "qr.png")
Если хотите считать с камеры, то напишите "Cam"
qr.png
```

Происходит переход на сайт



Пример 5 (считывание с картинки):

```
Hello!!
QR-код создан
Read
Чтобы считать QR-код с картинки напишите название этой картинки.(Например "qr.png")
Если хотите считать с камеры, то напишите "Cam"
qr.png
QR code message--> Hello!!
```

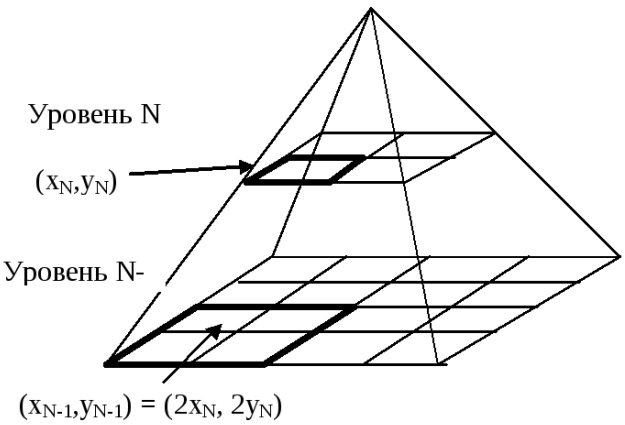
5.5. Использование пирамиды изображений для уменьшения разрешения видео

Описание алгоритма.

Пирамида изображений — это совокупность изображений, возникающих из одного исходного изображения, которые последовательно подвергаются субдискретизации до тех пор, пока не будет достигнута желаемая точка остановки.

В данной работе была рассмотрена пирамида Гаусса, которая используется для уменьшения разрешения изображений.

Пирамида представлена как набор слоев. При этом, чем выше слой, тем меньше размер и разрешение.



Чтобы получить слой (n + 1) в пирамиде Гаусса необходимо сделать следующее:

1. Свернуть предыдущий слой.
2. Удалите все четные строки и столбцы.

3. Повторение этого процесса над входным изображением создает всю пирамиду.

Можно легко заметить, что получившееся изображение будет иметь размер ровно в четверть площади предыдущего. Описанный выше алгоритм используется для уменьшения разрешения и размера изображения. Если же мы хотим увеличить размер, то добавляются строки и столбцы, которые заполняются нулями. При этом разрешение увеличить не получится, так как при уменьшении изображения дополнительная информация была потеряна.

Использованные функции

def createImagePyramid(imgName) – функция построения пирамиды изображений. Внутри были использованы следующие функции библиотеки OpenCV:

Функция	Описание
cv.imread(cv.samples.findFile(filename))	Загружает изображения из файла
cv.imshow('Kurs: image pyramid.', src)	Отображает изображение в окне
cv.waitKey(x)	Ожидает x секунда нажатия клавиши
cv.pyrUp(src, dstsize=(2 * cols, 2 * rows))	Увеличивает размер изображения без увеличения разрешения
cv.pyrDown(src, dstsize=(cols // 2, rows // 2))	Размывает изображение и уменьшает его разрешение
cv.destroyAllWindows()	Закрывает (уничтожает) все открытые программой окна

Внутри основного блока были использованы следующие функции библиотеки OpenCV:

Функция	Описание
cv.waitKey(x)	Ожидает x секунда нажатия клавиши
cv.imwrite(imgName, frame)	Сохраняет изображение

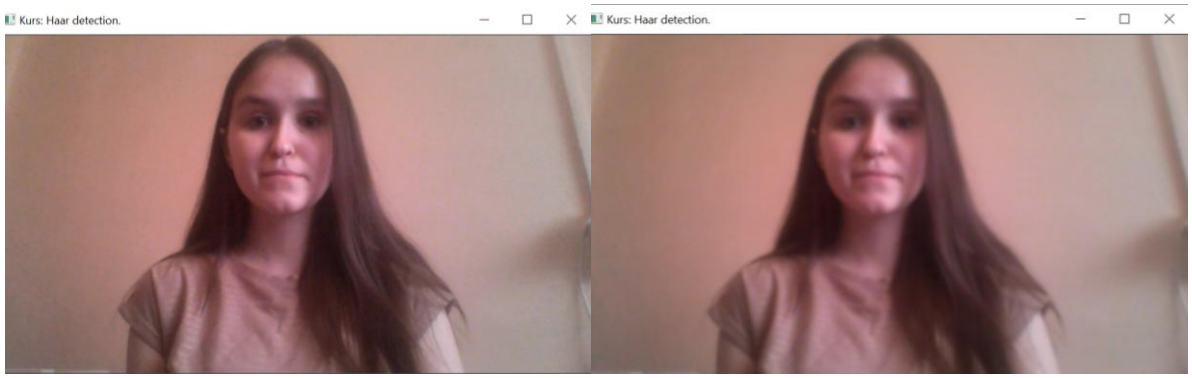
Внутри основного блока были использованы следующие функции библиотеки Argparse:

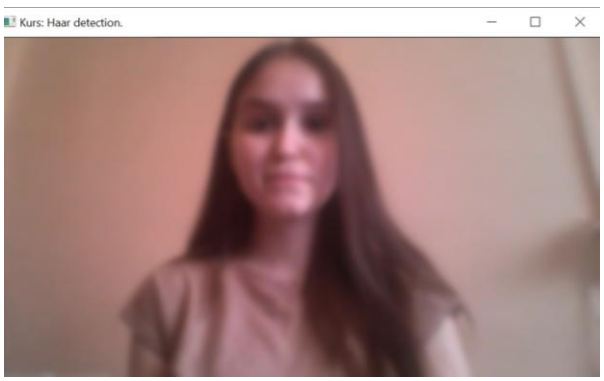
Функция	Описание
parser.add_argument('--camera', help='Camera number:', type=int, default=0)	Заполняет поля объекта класса ArgumentParser информацией об аргументах программы
parser.parse_args()	Анализирует аргументы объекта класса ArgumentParser

```
from __future__ import print_function
import cv2 as cv
import argparse

def createImagePyramid(imgName):
    print("""
    Demonstration of magnification ([i]) and reduction([0])
    ------------------
    * [m] -> Magnification
    * [r] -> Reduction
    * [ESC] -> Close our program
    """)

    filename = imgName
    #загрузка изображения
    src = cv.imread(cv.samples.findFile(filename))

    #проверка корректности загрузки изображения
    if src is None:
        print ('Error while opening image!')
        return -1

    while True:
        rows, cols, _channels = map(int, src.shape)
        cv.imshow('Kurs: image pyramid.', src)
        #выход по esc
        k = cv.waitKey(0)
        if k == 27:
            break

        #увеличение изображения
        elif chr(k) == 'm':
            src = cv.pyrUp(src, dstsize=(2 * cols, 2 * rows))
            print ('Magnification: Image x 2')

        #уменьшение изображения
        elif chr(k) == 'r':
            src = cv.pyrDown(src, dstsize=(cols // 2, rows // 2))
            print ('Reduction: Image / 2')

    cv.destroyAllWindows()
    return 0

parser = argparse.ArgumentParser(description='Kurs: Haar detection.')
parser.add_argument('--camera', help='Camera number:', type=int, default=0)
args = parser.parse_args()
cameraDevice = args.camera
```

```
#чтение потока видео
cap = cv.VideoCapture(cameraDevice)

if not cap.isOpened:
    print('Error while opening video!')
    exit(0)

imgCounter = 0
while True:
    ret, frame = cap.read()
    if frame is None:
        print('No captured frame!')
        break

    #выход по esc
    k = cv.waitKey(10)
    if k == 27:
        break

    #фотография по нажатию на пробел
    elif k == 32:
        imgName = "facedetect{}.png".format(imgCounter)
        cv.imwrite(imgName, frame)
        print("{} written!".format(imgName))
        createImagePyramid(imgName)
        imgCounter += 1
```

Пример работы программы





5.6. Наложение моделей на изображение с веб-камеры с использованием технологий OpenCV и нейронных сетей (фреймворк dnn.py)

Наложение моделей способно упростить задачу определения контуров изображения. В данной работе файлы имеют расширение torch 7 (.t7), используется библиотека Torch7 для реализации нейронных сетей.

Фреймворк dnn.py

Основная возможность dnn заключается в загрузке и запуске нейронных сетей (inference). При этом модель может быть создана в любом из трех фреймворков глубокого обучения — Caffe, TensorFlow или Torch; способ ее загрузки и использования сохраняется независимо от того, где она была создана.

Для того, чтобы корректно реализовать сборку проекта необходимо предварительно установить git и компилятор C++. После чего можно заняться непосредственно сборкой:

make -j5 (Linux)

cmake --build . --config Release -- /m:5 (Windows)

Последовательность работы

1. Создаем массив моделей типа torch
2. Считываем модель
3. Начинаем преобразовывать изображение с камеры
4. Накладываем определенную модель
5. Вводим q - для выхода, n - для перехода к следующей модели, p – для возврата к предыдущей модели

Функции dnn.py, использованные в программе

readNetFromTorch()	Чтение модели в формате torch7
blobFromImage()	Создает 4-х мерный blob-объект из изображения. При необходимости изменяет размер и обрезает изображение

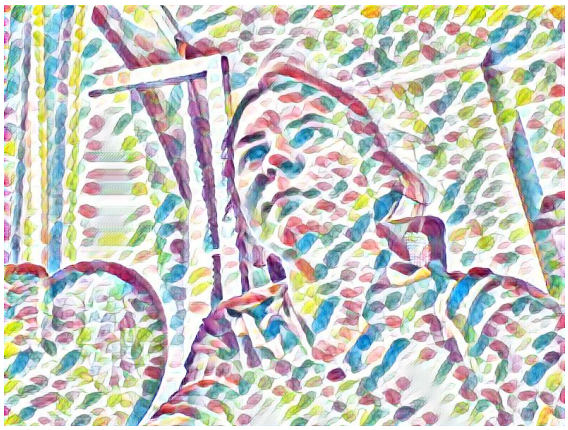
	от центра, вычитает средние значения, масштабирует значения по коэффициенту масштабирования, меняет местами синий и красный каналы
setInput()	Выводит blob-объект на экран

Листинг программы

```
import os
import cv2 as cv

brightness = (103.939, 116.779, 123.680)

def predict(img, h, w):
    # RGB settings
    blob = cv.dnn.blobFromImage(img, 1.0, (w, h), brightness, swapRB=False, crop=False)

    print('Set the image')
    net.setInput(blob)

    print('Start to transform')
    out = net.forward()
    print('Completing the transform')
    out = out.reshape((3, out.shape[2], out.shape[3]))
    out[0] += brightness[0]
    out[1] += brightness[1]
    out[2] += brightness[2]
    out /= 255.0
    out = out.transpose(1, 2, 0)
    return out


def resize_img(img, width=None, height=None, inter=cv.INTER_AREA):
    dim = None
    h, w = img.shape[:2]

    if width is None and height is None:
        return img
    elif width is None:
        r = height / float(h)
        dim = (int(w * r), height)
    elif height is None:
        r = width / float(w)
        dim = (width, int(h * r))

    resized = cv.resize(img, dim, interpolation=inter)
    return resized
models_path = './model/'
models = []
for f in sorted(os.listdir(models_path)):
```

```
    if f.endswith('.t7'):
        models.append(f)

path = models_path + ('' if models_path.endswith('/') else '/')
print(path + models[2])
model_loaded_i = -1
total_models = len(os.listdir(models_path))

model_loaded_i = 0
model_to_load = path + models[model_loaded_i]

net = cv.dnn.readNetFromTorch(model_to_load)
vid = cv.VideoCapture(0)
while True:
    _, frame = vid.read()
    img = resize_img(frame, width=800)
    h, w = img.shape[:2]
    out = predict(img, h, w)

    cv.imshow('Change style', out)

    key = cv.waitKey(1) & 0xFF
    if key == ord('q'):
        break
    elif key == ord('n'):
        model_loaded_i = (model_loaded_i + 1) % total_models
        model_to_load = path + models[model_loaded_i]
        net = cv.dnn.readNetFromTorch(model_to_load)
    elif key == ord('p'):
        model_loaded_i = (model_loaded_i - 1) % total_models
        model_to_load = path + models[model_loaded_i]
        net = cv.dnn.readNetFromTorch(model_to_load)

vid.release()
cv.destroyAllWindows()
```

Результаты работы




Результатом работы является «отредактированное» изображение с камеры, на которое накладываются модели с постоянным обновлением наложения.

5.7. Детектирование медицинской маски на лице

С 2020 года во многих странах введен масочный режим. В магазины пришлось нанять дополнительных сотрудников, которые пропускают только людей в масках. Каждого такого охранника можно заменить камеру, которая будет отслеживать наличие маски у человека. Задача состоит из 3 шагов. Первая шаг заключается в том, чтобы обучить нейронную сеть распознавать медицинскую маску на человеческом лице. Второй шаг заключается в поиске лица на потоке видео, для этого используется дэфолтная обученная модель на каскадах Хаара из библиотеки OpenCV. После того как на видео найдено лицо, к нему применяется модель из первого шага для определения наличия маски.

Шаг 1

1. Устанавливаем библиотеки: keras, cv2, sklearn, imutils, numpy
2. model = keras.Sequential(…) - Эта сеть свертки состоит из двух пар слоев Conv и MaxPool для извлечения объектов из набора данных. Затем следует слой Flatten и Dropout для преобразования данных в одномерный массив и обеспечения переобучения и дообучения модели.
3. model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc']) – для поиска оптимального общего алгоритма будем использовать метод Адамса (стохастический

градиентный спуск), почему его? Потому что метод легок в вычислениях и при этом показывает относительно быструю сходимость к верному результату.

4. Создаем папку train, в ней подпапки with_mask и without_mask и заносим туда 500 фото с маской и 500 без маски, соответственно. Чем больше фото, тем лучше. Аналогично создаем папку test с подпапками with_mask и without_mask и засовываем новые фотографии на которых будет проверяться наша обученная модель. ImageDataGenerator() – позволяет создать из 1 картинки много новых, путем поворота, изменения перспективы и других функций. Так мы можем получить в десятки раз больше фотографий, чем загрузили из интернета в папку train.
   Аналогичным образом расширяем набор тестовых данных.
5. ModelCheckpoint() - инициализируем контрольную точку обратного вызова, чтобы сохранять лучшую модель после каждой эпохи во время обучения.
6. model.fit_generator(… , epochs=10, …) – обучаем модель используя все настройки примененные в пунктах 2 и 3. epochs=10 означает, что мы 10 раз обучим нашу нейронную сеть на 1 и том же датасете. Нужно помнить, что мы используем ограниченный датасет, чтобы оптимизировать обучение и подстроить кривую под данные. Делается это с помощью градиентного спуска — итеративного процесса. Поэтому обновления весов после одного прохождения недостаточно

Шаг 2

Перейдем к коду 2.

С поиском лица все достаточно просто. В библиотеке OpenCV уже есть обученная модель, основанная на каскадах Хаара. Загрузим эту модель:

```
haarcascade =
cv2.CascadeClassifier('/home/user_name/.local/lib/python3.6/site-packages/cv2/data/haarcascade_frontalface_default.xml')
```

Затем, используя следующую функцию, будем получать «список координат с лицами на картинке»:

```
faces = haarcascade.detectMultiScale(rerect_size)
```

Шаг 3

Продолжение Кода 2

Теперь нам остается взять координаты лица, применить к ним обученный классификатор из первого шага, нанести рамочку на лицо и написать, есть маска или нет.

1. model=load_model("./model-010.h5") – загружаем нашу обученную модель модель.

2. В бесконечном цикле получаем кадры с камеры, находим на них лицо и к каждому лицу применяем:
   result=model.predict(reshaped)
   В result записывается 'without mask' или 'mask'
3. Затем рисуем прямоугольник вокруг лица
   cv2.rectangle(im,(x,y),(x+w,y+h),GR_dict[label],2) и подписываем есть маска или нет и какова точность определения.

Таблица используемых функций OpenCV

cv2.read()	Читает изображение из потока видео
cv2.resize()	Изменяет размер изображения
cv2. CascadeClassifier()	Загружает классификатор
cv2.rectangle()	Рисует прямоугольник
classifier.detectMultiScale()	Применяет классификатор к изображению
cv2.imshow()	Используется для отображения изображения в окне
cv2.waitKey()	Ждем до нажатия клавиши
cv2.destroyAllWindow()	Удаляем все открытые окна
cv2.putText()	Вставка изображения

Листинг программы

```
‘’’ 1 Код для обучения модели ‘’’

from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
import cv2 # opencv
from keras.models import Sequential
from keras.layers import Conv2D, Input, ZeroPadding2D, BatchNormalization, Activation,
MaxPooling2D, Flatten, Dense,Dropout
from keras.models import Model, load_model
from keras.callbacks import TensorBoard, ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.utils import shuffle
import imutils
import numpy as np
# установка параметров слоев для обучаемой модели
model = keras.Sequential([
Conv2D(100, (3,3), activation='relu', input_shape=(150, 150, 3)),
MaxPooling2D(2,2),
Conv2D(100, (3,3), activation='relu'),
MaxPooling2D(2,2),
```

```
Flatten(),
Dropout(0.5),
Dense(50, activation='relu'),
Dense(2, activation='softmax')
])
# установка метрики потерь и метода оптимизации
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
TRAINING_DIR = "./train" # тренировочный датасет

# генерация дополнительных изображений для увеличения датасета
train_datagen = ImageDataGenerator(rescale=1.0/255,
rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
fill_mode='nearest')
train_generator = train_datagen.flow_from_directory(TRAINING_DIR,
batch_size=10,
target_size=(150, 150))
VALIDATION_DIR = "./test" # тестовый датасет
validation_datagen = ImageDataGenerator(rescale=1.0/255)
validation_generator = validation_datagen.flow_from_directory(VALIDATION_DIR,
                batch_size=10,
                target_size=(150, 150))

# инициализируем контрольную точку обратного вызова, чтобы сохранять лучшую модель
# после каждой эпохи во время обучения
checkpoint = ModelCheckpoint('model2-{epoch:03d}.model', monitor='val_loss',verbose=0,
save_best_only=True,mode='auto')
# обучаем модель
history = model.fit_generator(train_generator,
                              epochs=10,
                              validation_data=validation_generator,
                              callbacks=[checkpoint])


‘’’ 2 Код демонстрирующий возможности обученной модели ‘’’

import cv2
import numpy as np
from keras.models import load_model

model=load_model("./model-010.h5") # Загружаем обученную модель
results={0:'without mask',1:'mask'}
GR_dict={0:(0,0,255),1:(0,255,0)}
rect_size = 4
cap = cv2.VideoCapture(0) # Захватываем изображение с камеры
```

```
haarcascade = cv2.CascadeClassifier('/home/user_name/.local/lib/python3.6/site-packages/cv2/data/haarcascade_frontalface_default.xml') # используем заранее обученную модель для поиска лица на видео
while True:
(rval, im) = cap.read()
im=cv2.flip(im,1,1) # отразить изображение
# уменьшаем размер изображения чтобы ускорить распознавание
rerect_size = cv2.resize(im, (im.shape[1] // rect_size, im.shape[0] // rect_size))
# находим лица на картинке
faces = haarcascade.detectMultiScale(rerect_size)
# рисуем прямоугольную рамку вокруг каждого лица
for f in faces:
(x, y, w, h) = [v * rect_size for v in f]
face_img = im[y:y+h, x:x+w] # сохраняем только рамки прямоугольника
rerect_sized=cv2.resize(face_img,(150,150))
normalized=rerect_sized/255.0
reshaped=np.reshape(normalized,(1,150,150,3))
reshaped = np.vstack([reshaped])
result=model.predict(reshaped)
label=np.argmax(result,axis=1)[0]
cv2.rectangle(im,(x,y),(x+w,y+h),GR_dict[label],2)
cv2.rectangle(im,(x,y-40),(x+w,y),GR_dict[label],-1)
cv2.putText(im, results[label], (x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)

      # Показ картинки
cv2.imshow('LIVE',  im)
key = cv2.waitKey(10)
if key == 27: # Завершение программы по клавише Esc
break
# остановить видеострим
cap.release()
# Закрыть все окна
cv2.destroyAllWindows()
```

Скриншоты результатов выполнения программы

5.8. Детектирование дорожных знаков на основе пороговой бинаризации

Детектировать – выделить область, в которой находится распознаваемый объект, на исходном изображении. Так, на примере ниже представлена картинка до детектирования и после:

Исходное изображение

Детектированный знак на изображении

Принцип работы детектирования по цвету

Одним из распространённых способов детектирования объектов является детектирование по цвету. Так, например, дорожные знаки имеют тот или иной цвет, с помощью которого их можно выделить на кадре. Однако, например, помимо синего знака на изображении также может находиться синяя машина того же цвета или иные объекты того же цвета, что и детектируемый. Поэтому после детектирования обязательно идет распознавание объекта, но уже не на всем изображении, а на детектированном участке.

Для того, чтобы произвести детектирование по цвету производят бинаризацию изображения (перевод изображения в 2 цвета – черное и белое).

В примерах ниже мы будем использовать бинаризацию по порогу. Так, например, для синего знака пороговым значением будет синий: все, что не синее будет черным, а все, что имеет синий цвет или похоже на него будет становится белым, тем самым образовывая контуры изображения.

После получения бинарного изображения мы находим все контуры белые области. Так как на изображении может быть помехи, берется только площадь самого большого контура. По окончании расчета наибольшего контура, мы вписываем его в горизонтально ориентированный прямоугольник и вырезаем из изображения часть по этому прямоугольнику.

Принцип распознавания сравнение с эталоном

Есть большое количество методов для распознавания объектов, в данном примере мы рассмотрим самый простой из них. Для сравнения результатов мы возьмем эталонное изображение и также проведем

бинаризацию по значению. Затем мы уменьшим эталонное изображение до 64 на 64 пикселя и попиксельно сравним их.

Практическая часть

Подбор пороговых значений бинаризации

Чтобы распознать знак бинаризацией по порогу, нам необходимо подобрать пороговые значения. Для этого разработаем небольшой алгоритм, который позволит нам увидеть какие цвета у нас разрешены, а какие запрещены, а также быстро менять пороговое значение.

Сначала создадим трекбары для изменения пороговых значений.

```
cv2.createTrackbar('minb', 'resultOfBnarization', 0, 255, nothing)
cv2.createTrackbar('ming', 'resultOfBnarization', 0, 255, nothing)
cv2.createTrackbar('minr', 'resultOfBnarization', 0, 255, nothing)

cv2.createTrackbar('maxb', 'resultOfBnarization', 0, 255, nothing)
cv2.createTrackbar('maxg', 'resultOfBnarization', 0, 255, nothing)
cv2.createTrackbar('maxr', 'resultOfBnarization', 0, 255, nothing)
```

Каждый кадр мы будем:

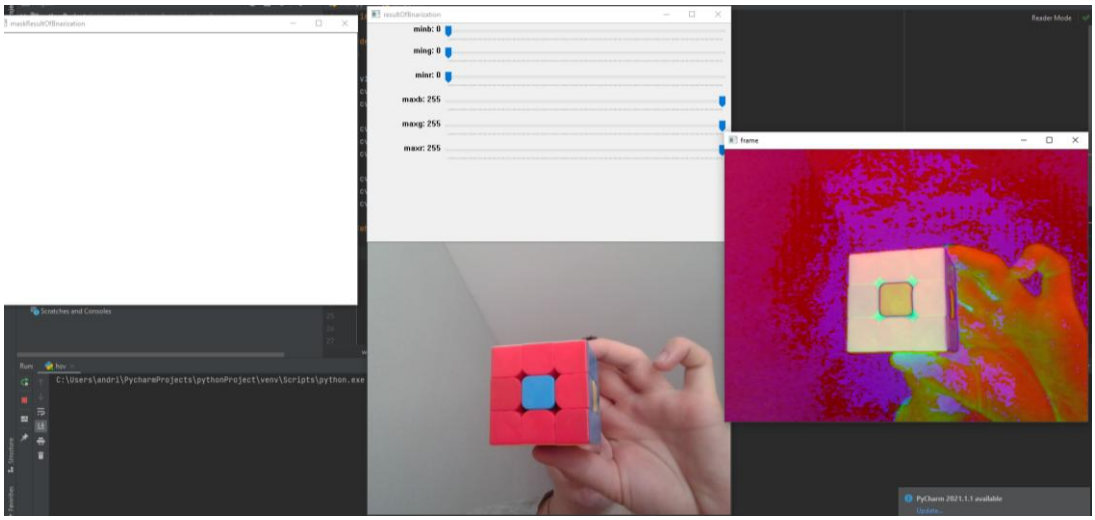
- Получать кадр через захват видео с камеры, конвертировать его в HSV цвета;
- Получать значения с ранее созданных трекбаров;
- Делать небольшой блюр цветов (для того, чтобы смягчить наши полученные изображения);
- Производить бинаризацией по порогу, используя метод inRange. Данная функция получает на вход кадр (в нашем случае в формате HSV), а также два вектора – верхнее значение и нижнее значение цвета. На выходе мы получаем изображение маску – все разрешенные цвета будут белыми, а все запрещенные цвета – черными.

```
_, frame = videoCapture.read();
hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
cv2.imshow('frame', hsv)
minb = cv2.getTrackbarPos('minb', 'resultOfBnarization')
ming = cv2.getTrackbarPos('ming', 'resultOfBnarization')
minr = cv2.getTrackbarPos('minr', 'resultOfBnarization')

maxb = cv2.getTrackbarPos('maxb', 'resultOfBnarization')
maxg = cv2.getTrackbarPos('maxg', 'resultOfBnarization')
maxr = cv2.getTrackbarPos('maxr', 'resultOfBnarization')

hsv = cv2.blur(hsv,(5,5))

mask = cv2.inRange(hsv, (minb, ming, minr), (maxb, maxg, maxr))
```

Для того, чтобы применить маску к исходному кадру используем функцию bitwise_and, которая принимает кадр и маску и на основе этого возвращает кадр, в котором остаются только те цвета, которые прощены маской.

```
result = cv2.bitwise_and(frame, frame, mask = mask)
```

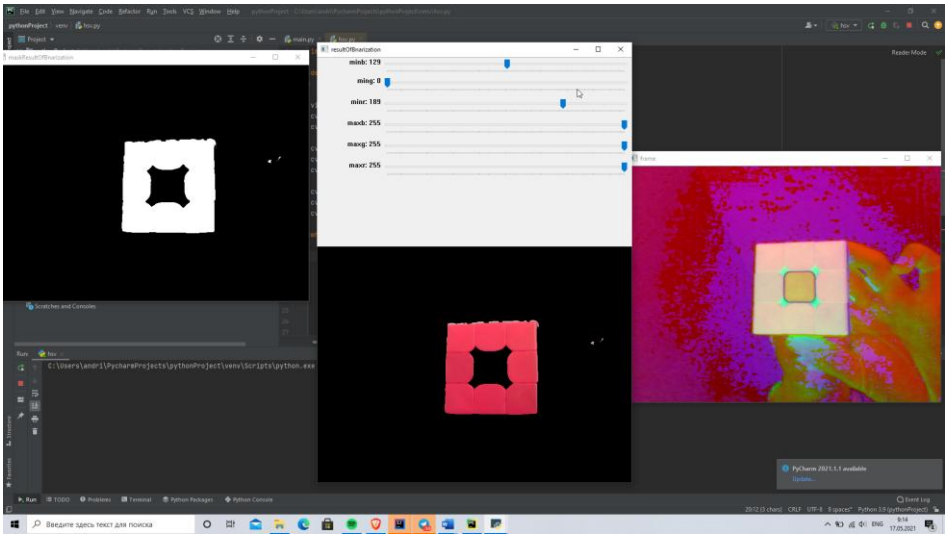
При запуске мы видим сразу несколько окон



Разберем их подробнее:

- В окне resultOfBnarization мы будем видеть созданные ранее трекпады, а также изображение, полученное в результате работы функций inRange и bitwise_and.
- В окне maskResultOfBnarization мы будем видеть маску нашего изображения,
- В окне frame мы будем видеть изображение в формате HSV.

При изменениях пороговых значений мы можем увидеть, как будет меняться маска и итоговое изображение. Так, например, поставим в качестве порога красный цвет кубика Рубика.



Полученные пороговые значения мы запоминаем, чтобы использовать их дальше при детектировании объекта.

Реализация алгоритма детектирования

После того, как мы подобрали пороговое значение приступим к детектированию.

Первым шагом, как и в скрипте для подбора пороговых значений, мы производим конвертацию в HSV, а также проводим бинаризацию по порогу (вектора для inRange берутся из этапа подбора пороговых значений) и смягчаем изображение. Для этого используем функции erode и dilate.

```
hsvFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
hsvFrame = cv2.blur(hsvFrame, (5, 5))
frameMask = cv2.inRange(hsvFrame, (56, 128, 188), (255, 255, 255))
cv2.imshow("Frame mask", frameMask)
frameMask = cv2.erode(frameMask, None, iterations=2)
frameMask = cv2.dilate(frameMask, None, iterations=4)
cv2.imshow("Smooth frame mask", frameMask)
```

Следующим этапом будет выделение контуров. Для этого используем функцию findContours, которая возвращает контуры на основе маски, сделанной ранее.

```
targetContours = cv2.findContours(frameMask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
targetContours = targetContours[0]
```

Если удалось найти хоть какие-то контуры, то мы выбираем самый большой из них. Для этого сортируем массив контуров, выбираем самый большой из них, чтобы отсеять погрешности. Затем мы вырезаем контур, на котором удалось детектировать контуры и переводим полученное изображение к размеру 64 на 64. Уменьшение изображения нужно для того, чтобы упростить дальнейшее сравнение по пикселям. В данном случае мы балансируем между точностью и скоростью вычислений.

```
if targetContours:
    targetContours = sorted(targetContours, key=cv2.contourArea, reverse=True)
    cv2.drawContours(frame, targetContours, 0, (0,0,255), 3)
    cv2.imshow("Contours of target", frame)
    (x,y, w, h) = cv2.boundingRect(targetContours[0])
    cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0), 2)
    cv2.imshow("Rect on frame", frame)
    cuttingFrame = frameForCutting[y:y+h, x:x+w]
    cuttingFrame = cv2.resize(cuttingFrame, (64, 64))
    cuttingFrame = cv2.inRange(cuttingFrame, (56, 128, 188), (255, 255, 255))
    cv2.imshow("Current image", cuttingFrame)
```

Последним этапом будет сравнение детектированного знака с эталонным изображением. Оба изображения заранее были обработаны функцией inRange на более ранних этапах.

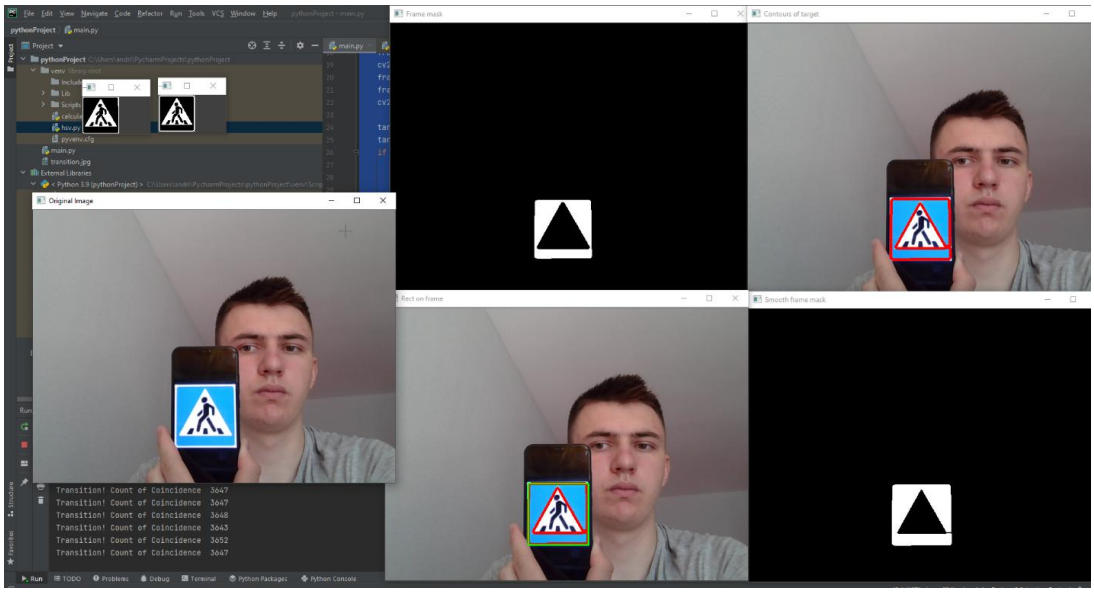
Для распознавания знака мы сравниваем по пикселям два изображения. Если количество совпадений будет больше 3100 (это значение можно подобрать на практике), то мы будем считать знак распознанным.

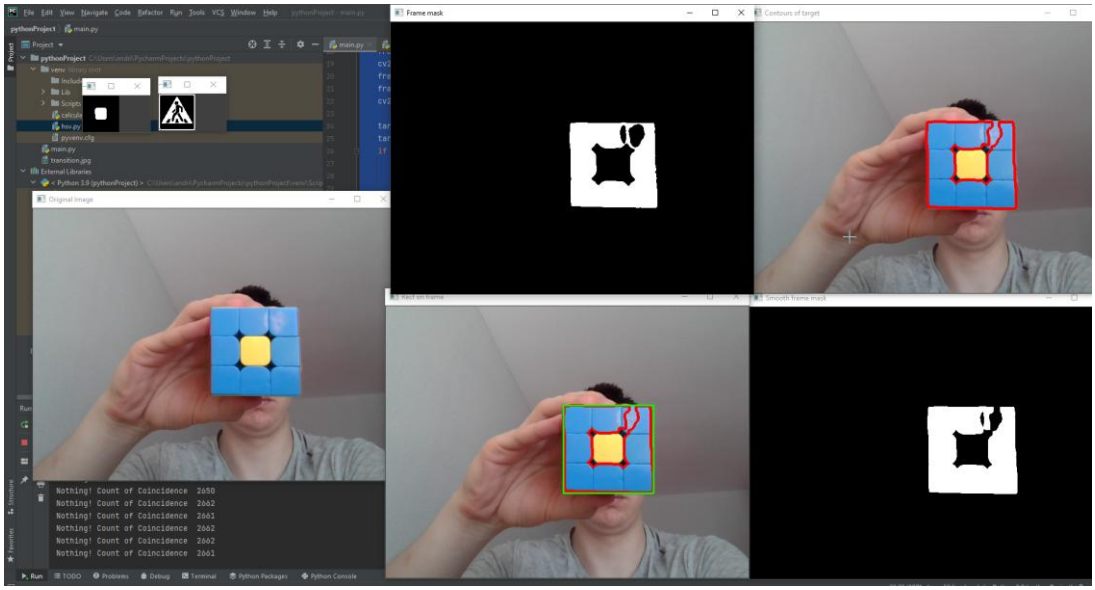
```
transitionCoincidence = 0
for h in range(64):
    for w in range(64):
        if cuttingFrame[h][w] == target[h][w]:
            transitionCoincidence += 1

if transitionCoincidence > 3100:
    print("Transition! Count of Coincidence ", transitionCoincidence)
else:
    print("Nothing! Count of Coincidence ", transitionCoincidence)
```

Примеры работы программы

При выполнении программы мы видим:

- Оригинальное (исходное) изображение
- Обычная маска в результате работы inRange
- Маска, смягченная функциями erode и dilate
- Контур детектированного изображения
- Наибольший контур, выбранный для распознавания
- Исходное изображение (справа) и изображение, полученное в результате детектирования (слева)





Результат работы можно увидеть в консоли. Так, например, в первом случае мы передаем знак и видим в консоли "Transition! Count of Coincidence 3652", что означает, что был расположен знак, а количество совпадний 3652 (при пороге 3100), что достаточно, чтобы однозначно сказать, что полученное и ожидаемое изображения совпали. Во втором же случае мы передаем знак и видим в консоли "Nothing! Count of Coincidence 2664", что означает, что не был расположен знак, а количество совпадний 2664 (при пороге 3100), чего недостаточно, чтобы однозначно сказать, что полученное и ожидаемое изображения совпали.

Программа подбора пороговых значений

```
import cv2

def nothing(x):
    pass

videoCapture = cv2.VideoCapture(0)
cv2.namedWindow('resultOfBnarization')
cv2.namedWindow('maskResultOfBnarization')

cv2.createTrackbar('minb', 'resultOfBnarization', 0, 255, nothing)
cv2.createTrackbar('ming', 'resultOfBnarization', 0, 255, nothing)
cv2.createTrackbar('minr', 'resultOfBnarization', 0, 255, nothing)

cv2.createTrackbar('maxb', 'resultOfBnarization', 0, 255, nothing)
cv2.createTrackbar('maxg', 'resultOfBnarization', 0, 255, nothing)
cv2.createTrackbar('maxr', 'resultOfBnarization', 0, 255, nothing)

while(True):
        _, frame = videoCapture.read();
```

```
        hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
        cv2.imshow('frame', hsv)
        minb = cv2.getTrackbarPos('minb', 'resultOfBnarization')
        ming = cv2.getTrackbarPos('ming', 'resultOfBnarization')
        minr = cv2.getTrackbarPos('minr', 'resultOfBnarization')

        maxb = cv2.getTrackbarPos('maxb', 'resultOfBnarization')
        maxg = cv2.getTrackbarPos('maxg', 'resultOfBnarization')
        maxr = cv2.getTrackbarPos('maxr', 'resultOfBnarization')

        hsv = cv2.blur(hsv,(5,5))

        mask = cv2.inRange(hsv, (minb, ming, minr), (maxb, maxg, maxr))

        cv2.imshow('maskResultOfBnarization', mask)
        result = cv2.bitwise_and(frame, frame, mask = mask)
        cv2.imshow('resultOfBnarization', result)

        if cv2.waitKey(1) == ord("q"):
            break

videoCapture.release()
cv2.destroyAllWindows()
```

Программа для детектирования дорожных знаков

```
import cv2


target = cv2.imread('transition.jpg')
target = cv2.resize(target, (64, 64))
target = cv2.inRange(target, (56, 128, 188), (255, 255, 255))
cv2.imshow("Target image", target)


videoCapture = cv2.VideoCapture(0)
while (True):
    _,frame = videoCapture.read()
    cv2.imshow("Original Image", frame)
    frameForCutting = frame.copy()

    hsvFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    hsvFrame = cv2.blur(hsvFrame, (5, 5))
    frameMask = cv2.inRange(hsvFrame, (56, 128, 188), (255, 255, 255))
    cv2.imshow("Frame mask", frameMask)
    frameMask = cv2.erode(frameMask, None, iterations=2)
    frameMask = cv2.dilate(frameMask, None, iterations=4)
    cv2.imshow("Smooth frame mask", frameMask)

    targetContours = cv2.findContours(frameMask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
    targetContours = targetContours[0]
    if targetContours:
        targetContours = sorted(targetContours, key=cv2.contourArea,
reverse=True)
        cv2.drawContours(frame, targetContours, 0, (0,0,255), 3)
        cv2.imshow("Contours of target", frame)
        (x,y, w, h) = cv2.boundingRect(targetContours[0])
        cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0), 2)
        cv2.imshow("Rect on frame", frame)
        cuttingFrame = frameForCutting[y:y+h, x:x+w]
        cuttingFrame = cv2.resize(cuttingFrame, (64, 64))
```

```
        cuttingFrame = cv2.inRange(cuttingFrame, (56, 128, 188), (255, 255,
255))
        cv2.imshow("Current image", cuttingFrame)

        transitionCoincidence = 0
        for h in range(64):
            for w in range(64):
                if cuttingFrame[h][w] == target[h][w]:
                    transitionCoincidence += 1

        if transitionCoincidence > 3100:
            print("Transition! Count of Coincidence ", transitionCoincidence)
        else:
            print("Nothing! Count of Coincidence ", transitionCoincidence)

    if cv2.waitKey(1) == ord("q"):
        break

videoCapture.release()
cv2.destroyAllWindows()
```

5.9. Определение дорожного знака

Описание Алгоритма. Алгоритм может распознавать только следующие знаки: движение прямо, движение прямо и налево, движение прямо и направо, движение направо, движение налево, знак стоп.

1. Преобразуем картинку в серый цвет для более быстрого распознавания круга

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

2. Размываем весь кадр, чтобы не обнаружить лишний круг

```
image = cv2.medianBlur(gray, 37)
```

3. Далее используем встроенный алгоритм обнаружения круга minDist = 50 это расстояние между центрами обнаруженных кругов, благодаря ему у нас не будет кругов находящихся в одном месте Из-за слишком большого этого значения мы можем не обнаружить некоторые круги param1 увеличивает количество обнаруженных кругов

```
circles = cv2.HoughCircles(image, cv2.HOUGH_GRADIENT, 1, 100, param1=120, param2=40)
```

param2 отбрасывает ложные круги.

4. Отфильтровываем самый большой круг, далее будем обнаружать знаки, кроме тех которые стоят вплотную к камере. Эта проверка предотвращает сбой программы при попытке индексировать список вне его диапазона. По факту просто выделяем квадрат на объекте, и делаем круг внутри

```
for i in range(len(circles[:, :, 2][0])):
    if circles[:, :, 2][0][i] > max_r and circles[:, :, 2][0][i] > 50:
        max_r = circles[:, :, 2][0][i]
        max_i = i

x, y, r = circles[:, :, :][0][max_i]

if x > r and y > r:
```

5. Проверяем много ли красного цвета на объекте, если да, то это знак стоп

```
    if main_color[2] > 100:
        print("Знак Стоп")
```

6. Проверка на синий цвет

```
elif main_color[0] > 80:
    # Делим квадрат на 3 зоны, и в них определяем преобладающий цвет

    zone_0 = square[square.shape[0] * 3 // 8:square.shape[0]
                    * 5 // 8, square.shape[1] * 1 // 8:square.shape[1] * 3 // 8]
    zone_0_color = get_main_color(zone_0, 1)

    zone_1 = square[square.shape[0] * 1 // 8:square.shape[0]
                    * 3 // 8, square.shape[1] * 3 // 8:square.shape[1] * 5 // 8]
    zone_1_color = get_main_color(zone_1, 1)

    zone_2 = square[square.shape[0] * 3 // 8:square.shape[0]
                    * 5 // 8, square.shape[1] * 5 // 8:square.shape[1] * 7 // 8]
    zone_2_color = get_main_color(zone_2, 1)
    if zone_1_color[2] < 60:
        if sum(zone_0_color) > sum(zone_2_color):
            print("Знак Движение влево")
        else:
            print("Знак Движение вправо")
    else:
        if sum(zone_1_color) > sum(zone_0_color) and sum(zone_1_color) > sum(zone_2_color):
            print("Знак Движение прямо")
        elif sum(zone_0_color) > sum(zone_2_color):
            print("Знак Движение прямо или налево")
        else:
            print("Знак Движение прямо или направо")
else:
    print("Нет знака")
```

7. Рисуем все обнаруженные круги на экране

```
    for i in circles[0, :]:
        cv2.circle(frame, (i[0], i[1]), i[2], (0, 255, 0), 2)
        cv2.circle(frame, (i[0], i[1]), 2, (0, 0, 255), 3)
cv2.imshow('camera', frame)
```

Описание работы функции get_main_color:

1. Применяем кластеризацию k-средних, чтобы создать палитру с наиболее репрезентативными цветами изображения

```
pixels = np.float32(image).reshape((-1, 3))
criterion = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 200, .1)
flags = cv2.KMEANS_RANDOM_CENTERS
flags, labels, centroids = cv2.kmeans(
    pixels, n_colors, None, criterion, 10, flags)
palette = np.uint8(centroids)
```

2. Определяем преобладающий цвет в палитре, который больше всех встречается на квантовом изображении

```
return palette[np.argmax(itemfreq(labels)[:, -1])]
```

Использованные функции

np.float32	Создание объекта типа float 32 bit
np.reshape	Придает новую форму массиву без изменения его данных.
`cv2.TERM_CRITERIA_EPS`	остановить итерацию алгоритма при достижении заданной точности epsilon.
`cv2.TERM_CRITERIA_MAX_ITER`	остановить алгоритм после указанного количества итераций, max_iter.
cv2.KMEANS_RANDOM_CENTERS	Если этот флаг включен, метод всегда начинается со случайного набора начальных выборок и пытается сойтись оттуда в зависимости от вашего TermCirteria
cv2.kmeans()	Кластеризация k-средних
np.argmax	Определение наибольшего аргумента для классов из NumPy
scipy.stats.itemfreq	Возврат двумерной таблицы частот
cv2.VideoCapture	Класс для захвата видео через файл, вебкамеру и др.
Cv2.waitkey()	функция привязки клавиатуры. Его аргумент - время в миллисекундах. Функция ожидает в течение указанных миллисекунд любого события клавиатуры
cv2.cvtColor	Преобразование цвета на картинке
cv2.medianBlur	Размытие кадра
cv2.HoughCircles	Встроенный алгоритм распознавания кругов
Np.uint16	Тип int 16 бит
Np.around	Равномерное округление до заданного порядка
cv2.circle	Рисование круга

cv2.imshow	Отображение картинки
cv2.VideoCapture.release()	Освобождение аппаратных ресурсов и вывод результаты программы

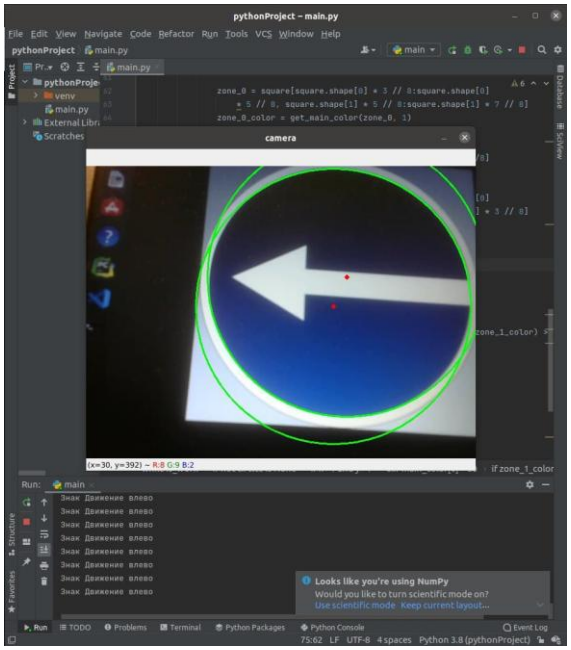
Листинг программы

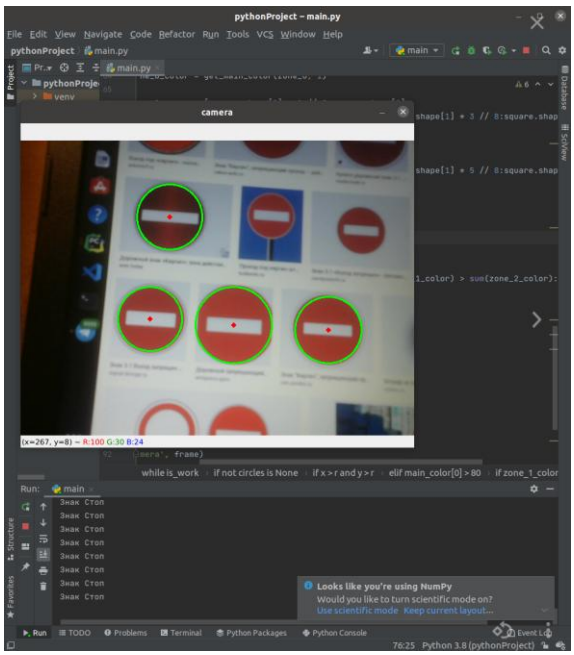
```
import cv2
import numpy as np
from scipy.stats import itemfreq
def get_main_color(image, n_colors):
    pixels = np.float32(image).reshape((-1, 3))
    criterion = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 200, .1)
    flags = cv2.KMEANS_RANDOM_CENTERS
    flags, labels, centroids = cv2.kmeans(
        pixels, n_colors, None, criterion, 10, flags)
    palette = np.uint8(centroids)
    return palette[np.argmax(itemfreq(labels)[:, -1])]

cameraCapture = cv2.VideoCapture(0)
cv2.namedWindow('camera')

is_work, frame = cameraCapture.read()

while is_work:
    cv2.waitKey(1)
    is_work, frame = cameraCapture.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    image = cv2.medianBlur(gray, 37)

    circles = cv2.HoughCircles(image, cv2.HOUGH_GRADIENT, 1, 100, param1=120, param2=40
)


    if not circles is None:
        max_r, max_i = 0, 0
        circles = np.uint16(np.around(circles))

        for i in range(len(circles[:, :, 2][0])):
            if circles[:, :, 2][0][i] > max_r and circles[:, :, 2][0][i] > 50:
                max_r = circles[:, :, 2][0][i]
                max_i = i

        x, y, r = circles[:, :, :][0][max_i]

        if x > r and y > r:
            square = frame[y-r:y+r, x-r:x+r]
            main_color = get_main_color(square, 2)
```

```
            if main_color[2] > 100:
                print("Знак Стоп")

            elif main_color[0] > 80:

                first_zone = square[square.shape[0] * 3 // 8:square.shape[0]
                                                             * 5 // 8, square.shape[1] * 1
// 8:square.shape[1] * 3 // 8]
                first_zone_color = get_main_color(first_zone, 1)

                second_zone = square[square.shape[0] * 1 // 8:square.shape[0]
                                                             * 3 // 8, square.shape[1] * 3
// 8:square.shape[1] * 5 // 8]
                second_zone_color = get_main_color(second_zone, 1)

                third_zone = square[square.shape[0] * 3 // 8:square.shape[0]
                                                             * 5 // 8, square.shape[1] * 5
// 8:square.shape[1] * 7 // 8]
                third_zone_color = get_main_color(third_zone, 1)
                if second_zone_color[2] < 60:
                    if sum(first_zone_color) > sum(third_zone_color):
                        print("Знак Движение влево")
                    else:
                        print("Знак Движение вправо")
                else:
                    if sum(second_zone_color) > sum(first_zone_color) and sum(second_zo
ne_color) > sum(third_zone_color):
                        print("Знак Движение прямо")
                    elif sum(first_zone_color) > sum(third_zone_color):
                        print("Знак Движение прямо или налево")
                    else:
                        print("Знак Движение прямо или направо")
            else:
                print("Нет знака")

        for i in circles[0, :]:
            cv2.circle(frame, (i[0], i[1]), i[2], (0, 255, 0), 2)
            cv2.circle(frame, (i[0], i[1]), 2, (0, 0, 255), 3)
    cv2.imshow('camera', frame)


cv2.destroyAllWindows()
cameraCapture.release()
```

Примеры работы программы:

Определение знака поворота налево Определение нескольких знаков Стоп





5.10. Распознавание автомобильных номеров

Требуется написать программу для распознавания автомобильных номеров с картинки и вывода их на экран в текстовом виде.

Для написания программы использовалась среда PyCharm и язык программирования Python. Также использовались библиотека компьютерного зрения OpenCV и библиотека для распознавания текста pytesseract. Так как библиотека pytesseract хорошо преобразует в текст только готовые и отлаженные изображения, работа была разделена на несколько частей:

1) Преобразовать исходное изображение в черно-белый формат
2) Преобразовать черно-белое изображение пороговой обработкой для упрощения нахождения контуров
3) Найти контуры изображения
4) Для каждого найденного контура выполнить процедуру преобразования в текст
5) Отсечь все «мусорные» тексты

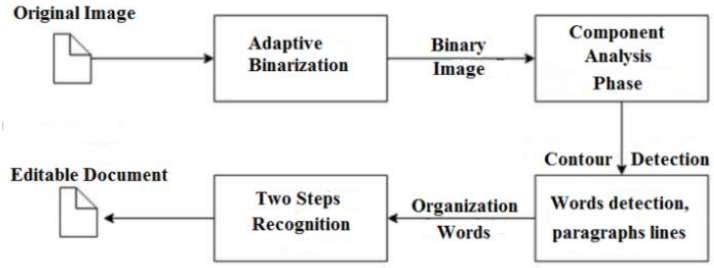
Исходное изображение:

Черно-белое изображение:

Изображение после пороговой обработки:

В заключительной части программы была использована функция распознавания текста image_to_string из библиотеки pytesseract. Tesseract был разработан на основе модели OCRopus на Python, которая была ответвлением LSMT на C ++ под названием CLSTM. CLSTM - это реализация модели рекуррентной нейронной сети LSTM на C ++ с использованием библиотеки Eigen для численных вычислений. Нейронная

сеть обучается на специальном файле rus.traineddata, который необходимо добавить в скачанную директорию библиотеки для корректной работы.



Поиск слов осуществлялся путем организации текстовых строк в капли, а строки и области анализировались на предмет фиксированного шага или пропорционального текста. Текстовые строки разбиваются на слова по-разному в зависимости от интервала между символами. Затем распознавание происходит в два этапа. На первом проходе делается попытка распознать каждое слово по очереди. Каждое удовлетворительное слово передается в адаптивный классификатор в качестве обучающих данных. Затем адаптивный классификатор получает возможность более точно распознавать текст внизу страницы.

Как видно по результату пороговой обработки, контур номера легко находится и текст распознается. В итоге программа выводит в консоль следующий текст:

```
"E207кх177.
```

Таблица используемых функций

Название	Применение
cv2.threshold(a, b, c)	Функция пороговой обработки. возвращает изображение, в котором все пиксели, которые темнее (меньше) a заменены на b, а все, которые ярче (больше) a, — на c.
cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)	Функция цветовой сегментации и нахождения контуров изображения по иерархии их вхождений. Также используется аппроксимация для сглаживания контуров.
contours.sort_contours	Сортировка контуров
pytesseract.image_to_string	Функция распознавания текста на изображении

Листинг программы

```
import cv2
import pytesseract
from imutils import contours

pytesseract.pytesseract.tesseract_cmd =
'C:\\Users\\nikit\\AppData\\Local\\Programs\\Tesseract-OCR\\tesseract.exe'

image = cv2.imread("car.jpg")

height, width, _ = image.shape
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # преобразовать изображение в
черно-белое
cv2.imshow("Gray Image", image_gray) #вывести на экран для проверки
cv2.waitKey()

threshold = cv2.threshold(image_gray, 0, 255, cv2.THRESH_OTSU)[1] # пороговая
обработка для выделения контуров
cv2.imshow("Threshold", threshold) #вывести на экран для проверки
cv2.waitKey()

contourss = cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) #
Найдем все контуры

contourss, _ = contours.sort_contours(contourss[0]) # отсортируем их

chars = set('0123456789,') # создадим множество цифр для отсечения "мусорных" текстов
for c in contourss: # переберем все контуры
     area = cv2.contourArea(c) # найдем площадь контура
     x, y, w, h = cv2.boundingRect(c)
     if area > 5000: # если площадь соизмерима с номером
          img = image[y:y+h, x:x+w] # получим этот контур из исходного изображения
          result = pytesseract.image_to_string(img, lang='rus+eng') # преобразуем его
в текст
          if not (' ' in result) and len(result) > 7 and any((c in chars) for c in
result):
               # отсекая "мусорные" варианты, выведем распознанный номер в консоль
               while '\n' in result:
                    result = ('\n'.join(result.split('\n')[:-1])) # избавимся от
лишних переносов строки
               print(result)
cv2.waitKey()
```

5.11. Замена фона в потоковом видео

В программе с помощью библиотек OpenCV и NumPy была реализована функция удаления зелёного фона с потокового видео,

полученного с веб-камеры, и его последующая замена на изображение из пользовательского файла.

Алгоритм работы программы

1. Подключение необходимых библиотек
2. Захват видео с веб-камеры, открытие пользовательского файла, создание окна настроек, в котором пользователь будет корректировать распознавание зеленого фона.
3. Цикл до закрытия программы:
   3.1. Считывание очередного кадра с веб-камеры и пользовательского файла.
   3.2. Создание массивов пикселей для обоих видео.
   3.3. Конвертация массивов пикселей в схему HSV.
   3.4. Считывание значений ползунков настроек.
   3.5. Создание массивов для настроенных пользователем «Яркого зеленого» и «Темного зеленого».
   3.6. Конструирование масок.
   3.7. Удаление элементов из массивов пикселей в соответствии с масками.
   3.8. Объединение обработанного видео с веб-камеры и пользовательского видео.
   3.9. Вывод результата на экран.
4. Выход из программы осуществляется нажатием клавиши «Q»

Список используемых функций

VideoCapture	Захват потокового видео (веб-камера или файл)
namedWindow	Создание именованного окна
createTrackbar	Создание ползунка
getTrackbarPos	Считывание значения ползунка
VideoCapture.read	Считывание следующего кадра из потокового видео
cvtColor	Перевод изображения в другую цветовую схему
inRange	Отбор элементов, которые соответствуют настроенному зелёному
bitwise_not	Инвертирование картинки побитовым отрицанием
bitwise_and	Совмещение картинок побитовым «И»
add	Совмещение картинок

imshow	Вывод окна
waitKey	Ожидание нажатия клавиши завершения
VideoCapture.release	Отмена захвата потокового видео
destroyAllWindows	Закрытие всех открытых окон

NumPy

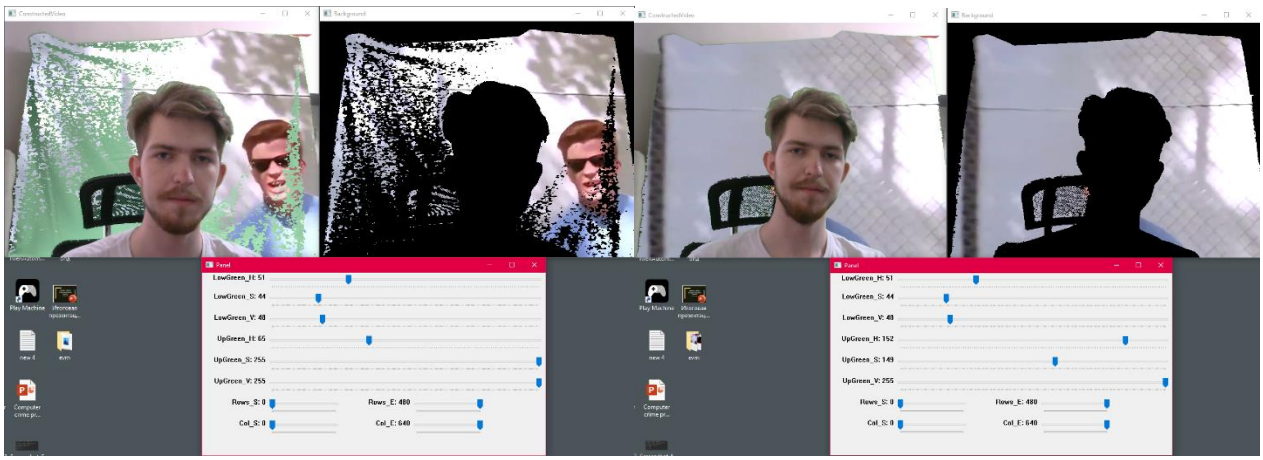
array	Создание массива

Листиг программы

```
import cv2
import numpy as np

if __name__ == '__main__':
    cap = cv2.VideoCapture(0)
    bg_vid = cv2.VideoCapture('bg.mp4')
    panel = np.zeros([100, 700], np.uint8)
    cv2.namedWindow('Panel')
    def nothing(x):
        pass
    cv2.createTrackbar('LowGreen_H', 'Panel', 0, 179, nothing)
    cv2.createTrackbar('LowGreen_S', 'Panel', 0, 255, nothing)
    cv2.createTrackbar('LowGreen_V', 'Panel', 0, 255, nothing)
    cv2.createTrackbar('UpGreen_H', 'Panel', 179, 179, nothing)
    cv2.createTrackbar('UpGreen_S', 'Panel', 255, 255, nothing)
    cv2.createTrackbar('UpGreen_V', 'Panel', 255, 255, nothing)
    cv2.createTrackbar('Rows_S', 'Panel', 0, 480, nothing)
    cv2.createTrackbar('Rows_E', 'Panel', 480, 480, nothing)
    cv2.createTrackbar('Col_S', 'Panel', 0, 640, nothing)
    cv2.createTrackbar('Col_E', 'Panel', 640, 640, nothing)
    while True:
        _, frame = cap.read()
        _, frameBG = bg_vid.read()

        s_r = cv2.getTrackbarPos('Rows_S', 'Panel')
        e_r = cv2.getTrackbarPos('Rows_E', 'Panel')
        s_c = cv2.getTrackbarPos('Col_S', 'Panel')
        e_c = cv2.getTrackbarPos('Col_E', 'Panel')

        roi = frame[s_r: e_r, s_c: e_c]
        bg_roi = frameBG[s_r: e_r, s_c: e_c]

        hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
        l_h = cv2.getTrackbarPos('LowGreen_H', 'Panel')
        u_h = cv2.getTrackbarPos('UpGreen_H', 'Panel')
        l_s = cv2.getTrackbarPos('LowGreen_S', 'Panel')
        u_s = cv2.getTrackbarPos('UpGreen_S', 'Panel')
```

```
        l_v = cv2.getTrackbarPos('LowGreen_V', 'Panel')
        u_v = cv2.getTrackbarPos('UpGreen_V', 'Panel')

        lower_green = np.array([l_h, l_s, l_v])
        upper_green = np.array([u_h, u_s, u_v])
        mask = cv2.inRange(hsv, lower_green, upper_green)
        mask_inv = cv2.bitwise_not(mask)

        bg = cv2.bitwise_and(bg_roi, bg_roi, mask=mask)
        fg = cv2.bitwise_and(roi, roi, mask=mask_inv)
        dst = cv2.add(bg, fg)

        cv2.imshow('Background', bg)
        cv2.imshow('ConstructedVideo', dst)

        cv2.imshow('Panel', panel)

        k = cv2.waitKey(30) & 0xFF
        if ord('q') == k:
            break

cap.release()
cv2.destroyAllWindows()
```

Скриншоты работы программы



5.12. Управление курсором мыши через жесты руки

В работе с помощью библиотек OpenCV, MediaPipe и AutoPy была реализована программа управления курсором мыши с помощью жестов руки. Так же реализована функция нажатия на левую кнопку мыши.

Алгоритм работы программы

1. Подключение нужных для работы программы библиотек:

a. OpenCV – захват и обработка потокового видео с веб камеры.
b. MediaPipe – определение и отслеживание положения рук и пальцев относительно изображения.
c. AutoPy – управление системами компьютера через различные команды.
d. Numpy, Math, Time – дополнительные библиотеки, используемые для вычислений.

2. Определение нужных для работы программы переменных.
3. Работа в цикле

a. Считывание изображения с камеры, перевод кодировки из BGR в RGB.
b. Нахождение значений координат точек соединений на кисти руки.
c. Отрисовка точек соединения на кисти руки, также дополнительно были выделены кончики указательного и большого пальцев.
d. Проверка поднятых пальцев руки.
   1. При поднятии только указательного пальца происходит перемещение курсора мыши.
   2. При одновременном поднятии сомкнутых друг к другу указательного и среднего пальцев происходит нажатие на ЛКМ. Также для нажатия следовало завести переменную задержки, чтобы нажатие на ЛКМ не было постоянным.
e. Вывод изображения на экран.

4. Для выхода из цикла и завершения программы следует нажать клавишу ESC.

Список используемых функций

VideoCapture	Захват видеокамеры
set	Выставление значений ширины и высоты камеры
read	Захват изображения с видеокамеры
cvtColor	Функция позволяющая переводить изображение из одной цветовой кодировки в другую
circle	Отрисовка круга или окружности

rectangle	Отрисовка прямоугольника
line	Отрисовка линии
imshow	Создание окна и показ изображения
waitKey	Ожидание пользовательского ввода

MediaPipe:

process	Определение кистей рук на изображении
draw_landmarks	Отрисовка связей между соединениями на кисти руки

AutoPy:

move	Перемещает курсор по заданным координатам
size	Возвращает рабочее разрешение экрана монитора
click	Нажатие клавиши мыши. По умолчанию нажимается ЛКМ

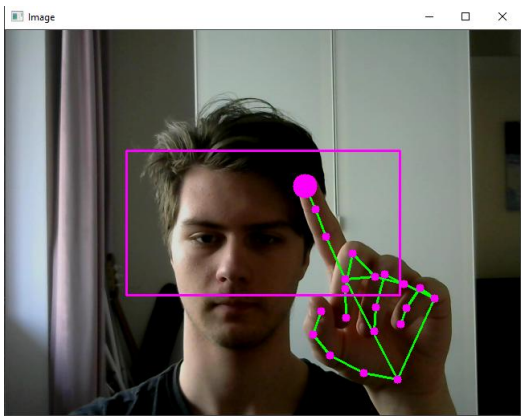
Листинг программы

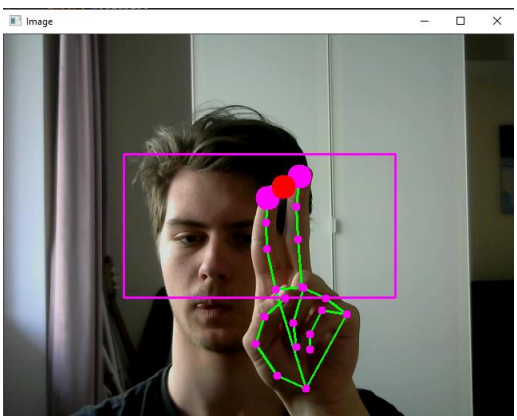
```
import cv2
import numpy
import mediapipe
import math
import time
import autopy

if __name__ == '__main__':
    cam_width, cam_height = 640, 480
    frame_rate = 150
    smoothening = 7

    prev_loc_X, prev_loc_Y = 0, 0
    curr_loc_X, curr_loc_Y = 0, 0

    camera = cv2.VideoCapture(0)
    camera.set(3, cam_width)
    camera.set(4, cam_height)
    hands = mediapipe.solutions.hands.Hands(False, 1)
    fingersTips = [4, 8, 12, 16, 20]

    width_scr, height_scr = autopy.screen.size()
    delayTime = 0
    clickCounter = 0

    while True:
        # Находим значения точек соединения на руке
        _, image = camera.read()
        imageRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        results = hands.process(imageRGB)
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                mediapipe.solutions.drawing_utils.draw_landmarks(image,
hand_landmarks, mediapipe.solutions.hands.HAND_CONNECTIONS)
```

```
        xList = []
        yList = []
        bbox = []
        landmarkList = []
        if results.multi_hand_landmarks:
            myHand = results.multi_hand_landmarks[0]
            for id, lm in enumerate(myHand.landmark):
                h, w, c = image.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                xList.append(cx)
                yList.append(cy)
                landmarkList.append([id, cx, cy])
                cv2.circle(image, (cx, cy), 5, (255, 0, 255), cv2.FILLED)
            xmin, xmax = min(xList), max(xList)
            ymin, ymax = min(yList), max(yList)
            bbox = xmin, ymin, xmax, ymax
        perTime = time.time()
        # Получаем координаты кончика указательного и среднего пальца
        if len(landmarkList) != 0:
            x1, y1 = landmarkList[8][1:]
            x2, y2 = landmarkList[12][1:]
            # Проверяем какие из пальцев подняты
            fingers = []
            if landmarkList[fingersTips[0]][1] > landmarkList[fingersTips[0] -
1][1]:
                fingers.append(1)
            else:
                fingers.append(0)
            for id in range(1, 5):
                if landmarkList[fingersTips[id]][2] < landmarkList[fingersTips[id]
- 2][2]:
                    fingers.append(1)
                else:
                    fingers.append(0)
            cv2.rectangle(image, (frame_rate, frame_rate), (cam_width -
frame_rate, cam_height - frame_rate), (255, 0, 255), 2)
            # Только указательный палец поднят
            if fingers[1] == 1 and fingers[2] == 0:
                # Переводим координаты для перемещения мыши
                x3 = numpy.interp(x1, (frame_rate, cam_width - frame_rate), (0,
width_scr))
                y3 = numpy.interp(y1, (frame_rate, cam_height - frame_rate), (0,
height_scr))
                # Сглаживание значений
                curr_loc_X = prev_loc_X + (x3 - prev_loc_X) / smoothening
                curr_loc_Y = prev_loc_Y + (y3 - prev_loc_Y) / smoothening

                # Перемещение мыши
                autopy.mouse.move(width_scr - curr_loc_X, curr_loc_Y)
```

```
                cv2.circle(image, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
                prev_loc_X, prev_loc_Y = curr_loc_X, curr_loc_Y

            if (delayTime > 0.001):
                # Указательный и средный палец подняты
                if fingers[1] == 1 and fingers[2] == 1:
                    # Находим расстояние между указательным и средним пальцем
                    # 8, 12, img
                    x1, y1 = landmarkList[8][1:]
                    x2, y2 = landmarkList[12][1:]
                    cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
                    cv2.line(image, (x1, y1), (x2, y2), (255, 0, 255), 3)
                    cv2.circle(image, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
                    cv2.circle(image, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
                    cv2.circle(image, (cx, cy), 15, (0, 0, 255), cv2.FILLED)
                    length = math.hypot(x2 - x1, y2 - y1)
                    lineInfo = [x1, y1, x2, y2, cx, cy]
                    # Одиночное нажатие мыши, если счетчик меньше 1 и задержка
пройдена
                    if (clickCounter < 1):
                        if length < 40:
                            cv2.circle(image, (lineInfo[4], lineInfo[5]),
                                       15, (0, 255, 0), cv2.FILLED)
                            autopy.mouse.click()
                            clickCounter += 1
                            delayTime = 0
                    else:
                        clickCounter = 0
            else:
                delayTime += time.time() - perTime
        # Вывод
        cv2.imshow("Image", image)
        # Закрыть, если нажата клавиша ESC
        if cv2.waitKey(1) == 27:
            exit(0)
```

Скриншоты работы программы





5.13. Игра аэрохоккей

Работа осуществляется с веб-камерой. Формируется мячик и 2 платформы. Одной из платформ управляем через веб-камеру. При контакте с границами игрового поля или с одной из платформ, мячик меняет траекторию передвижения и отскакивает, в зависимости от места столкновения. При выходе за боковые границы поля регистрируются забитые очки. Задача игрока набрать как можно большее количество очков.

Список используемых функций

Название	Краткое описание
cv2.cvtColor	Преобразует цветовое пространство матрицы
cv2.inRange	принимает три параметра: изображение, нижнее и верхнее значения диапазона. Она возвращает бинарную маску (ndarray из единиц и нулей) размером с оригинальное изображение. Единицы обозначают значения в пределах диапазона, а нули — вне его:
cv2.moments	Функция вернет нам массив моментов вплоть до третьего порядка.
cv2.circle	Рисует точку с указанными размерами и цветом, на указанных координатах.
cv2.destroyAllWindows	Уничтожает все открытые окна
cv2.imshow	Метод по отображению изображения в отдельном окне. Размер окна автоматически подстраивается под размер изображения

Пользовательские функции

- ball_animation() – отвечает за передвижения мяча по полю, а именно за изменение его траектории при столкновении с вертикальными границами поля или платформами.
- player_animation() – отвечает за передвижение платформы игрока, при достижении платформой вертикальных границ поля движение платформы останавливается, и может быть продолжено лишь в противоположном направлении
- opponent_animation() – функция отвечающая за передвижение платформы оппонента

- помимо ограничения на выход за пределы границ, добавлено отслеживание положения мяча на поле, с последующей корректировкой передвижения (если это необходимо).
- tip_off() – взбрасывание мяча, происходит в начале игры и после каждого забитого гола, внутри осуществляется выбор направления в котором мяч полетит от центра поря на этот раз
- camera_capture() – функция отвечающая за отслеживание зеленого (управляющего) цвета, когда камера находит зеленый цвет в достаточном количестве , появляется точка по центру зеленой области, координаты которой впоследствии будет перехватывать игра.

Листинг программы

```
import pygame
import sys
import random
import cv2
import numpy

def ball_animation():
    global ball_speed_x, ball_speed_y, player_score, opponent_score

    # Move animation
    ball.x += ball_speed_x
    ball.y += ball_speed_y
    # When ball out of window
    if ball.top <= 0 or ball.bottom >= screen_height:
        ball_speed_y *= -1

    if ball.left <= 0:
        player_score += 1
        tip_off()
    if ball.right >= screen_width:
        opponent_score += 1
        tip_off()

    # Condition for collide this player

    if ball.colliderect(player) or ball.colliderect(opponent):
        ball_speed_x *= -1

# When player go out of window
```

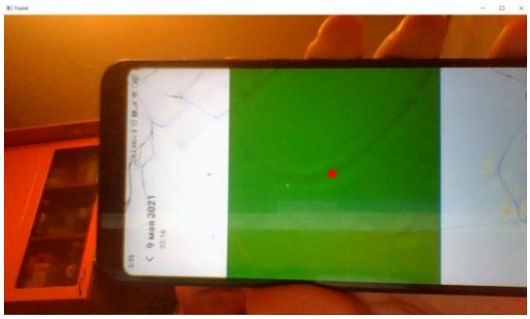
```
def player_animation():
    if player.top <= 0:
        player.top = 0
    if player.bottom >= screen_height:
        player.bottom = screen_height
def opponent_animation():
    # Opponent tracing ball
    if opponent.top < ball.y:
        opponent.top += opponent_speed_y
    if opponent.bottom > ball.y:
        opponent.bottom -= opponent_speed_y

    # When opponent go out of window
    if opponent.top <= 0:
        opponent.top = 0
    if opponent.bottom >= screen_height:
        opponent.bottom = screen_height

# When some one take a point, new one tip-off
def tip_off():
    global ball_speed_y, ball_speed_x

    ball.center = (screen_width / 2, screen_height / 2)
    ball_speed_y *= random.choice((1, -1))
    ball_speed_x *= random.choice((1, -1))

cap = cv2.VideoCapture(0)

cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

def camera_capture():
    global point_x, point_y, is_trace

    _, frame = cap.read()
    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    low_green = numpy.array([40, 40, 40])
    high_green = numpy.array([70, 255, 255])

    mask_green = cv2.inRange(hsv_frame, low_green, high_green)

    moments = cv2.moments(mask_green, 1)

    dM01 = moments['m01']
    dM10 = moments['m10']
    dArea = moments['m00']

    if dArea > 150:
```

```
        point_x = int(dM10 / dArea)
        point_y = int(dM01 / dArea)
        cv2.circle(frame, (point_x, point_y), 10, (0, 0, 255), -1)
        is_trace = 1
    else:
        is_trace = 0

    cv2.imshow("Frame", frame)

# General setup
pygame.init()
clock = pygame.time.Clock()

# Main window
screen_width = 1280
screen_height = 720

# Speed of animation
ball_speed_x = 15 * random.choice((1, -1))
ball_speed_y = 15 * random.choice((1, -1))
player_speed_y = 0
opponent_speed_y = 17
# Point cord
point_y = 360

# Tracking status
is_trace = 0

# Text on screen
player_score = 0
opponent_score = 0
game_font = pygame.font.Font("freesansbold.ttf", 32)

screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption('PONG')

# game Rectangles
ball = pygame.Rect(screen_width / 2 - 10, screen_height / 2 - 10, 20, 20)
player = pygame.Rect(screen_width - 20, screen_height / 2 - 70, 10, 140)
opponent = pygame.Rect(10, screen_height / 2 - 70, 10, 140)

# colors
green_color = (0, 255, 0)
red_color = (255, 0, 0)
blue_color = (0, 0, 255)
black_color = pygame.Color('black')
white_color = pygame.Color('white')

while True:
```

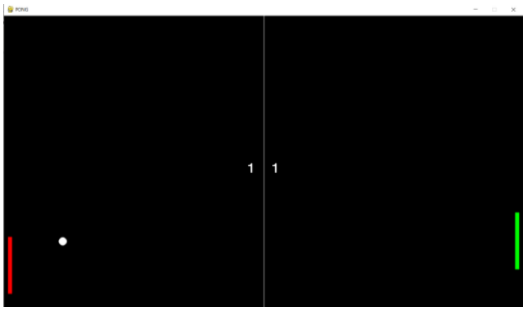
```
    # Handling window

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
    camera_capture()

    if is_trace == 1:
        ball_animation()
        player_animation()
        opponent_animation()
        player.y = point_y

    # draw objects
    screen.fill(black_color)
    pygame.draw.aaline(screen, white_color, (screen_width / 2, 0), (screen_width / 2,
screen_height))
    pygame.draw.rect(screen, green_color, player)
    pygame.draw.ellipse(screen, white_color, ball)
    pygame.draw.rect(screen, red_color, opponent)
    player_text = game_font.render(f"{player_score}", False, white_color)
    screen.blit(player_text, (660, 360))
    opponent_text = game_font.render(f"{opponent_score}", False, white_color)
    screen.blit(opponent_text, (600, 360))

    pygame.display.flip()
                                        clock.tick(60)
```

Скриншоты

Нахождение зеленого цвета и создание управляющей (красной) точки:



Процесс игры:


1 1

Заключение

В монографии рассмотрены некоторые из большого множества алгоритмов, которые применяются в компьютерном зрении. Эти алгоритмы и соответственно программы разрабатывались с участием студентов высшей школы программной инженерии СПбПУ. Они позволяют сделать первый шаг в области изучения программирования компьютерного зрения с использованием функций библиотеки OpenCV.

В монографии представлены примеры более 60 алгоритмов обработки статических изображений и видео, в которых используются функции OpenCV. Примеры программ написаны на языке Python - одного из самых распространенных языков программирования в областях компьютерного зрения и машинного обучения. В качестве среды разработки можно использовать PyCharm или другую IDE. Представленную работу можно рассматривать в качестве необходимого элемента в работе преподавателя в университете [41]. Задания для изучения и освоения некоторых алгоритмов представлены также в лабораторном практикуме [42], который может быть также полезен преподавателям и студентам.

Особенностью монографии является то, что в ней приводятся примеры реализаций не только традиционных алгоритмов обработки, но и другие примеры: видеоигры (аэрохоккей), нйронные сети, рисование, создание и чтение QR-кодов и др.

Данная монография будет полезна для специалистов в области проектирования и применения видеосистем для обработки сигналов и изображений. Она полезна студентам, проходящим подготовку по направлениям 09.03.01 «Информатика и вычислительная техника», 09.03.04 «Программная инженерия»,12.03.01 «Приборостроение».

Список литературы

1. R.Szeliski, "Computer Vision: Algorithms and Applications" http://szeliski.org/Book/ (В свободном доступе)
2. Цифровая обработка видеоизображений. А.А. Лукьяница, А.Г. Шишкин Ай-Эс-Эс Пресс 2009 512с ISBN: 978-5-9901899-1-1
3. James F Peters Foundations of Computer Vision. Publisher: Springer International Publishing Switzerland. 2017. 431 p. ISBN 978-3-319-30260-7, ISBN 978-3-319-30262-1 (eBook) DOI: 10.1007/978-3-319-52483-2
4. David Marr Vision: A computational investigation into the human representation and processing of visual information The MIT Press 2010 (originally published in 1982)
5. Молодяков С.А. Проектирование специализированных цифровых видеокамер. / СПб. : Изд-во Политехн. ун-та, 2016 .- 286 с. — ISBN 978-5-7422-5334-1
6. Молодяков С.А. Низкоуровневая обработка изображений в компьютерном зрении Санкт-Петербургский политехнический университет Петра Великого, Санкт-Петербург, 2021.- 197 с.
7. OpenCV. URL: http:// opencv.org/; URL: https://docs.opencv.org/
8. Joseph Howse, Joe Minichino Learning OpenCV 4 Computer Vision with Python 3, 3rd Edition 2020 372p
9. Программное обеспечение обработки изображений для цифровых камер Levenhuk http://www.levenhuk.ru/products/materials/0/levenhuk-ToupView-ru.pdf
10. Гуров И.П., Пименов А.Ю. Оценка влияния рассеянного излучения на качество формирования изображений в системах спектральной оптической когерентной томографии с электронным сканированием объектов // Оптический журнал -2020. - Т. 87. - № 7. - С. 18-23
11. Даееф Ф. Применение системы технического зрения при управлении мобильным роботом в динамической среде // Информатика,

телекоммуникации и управление. 2020. Т. 13. №1. С. 19-30. DOI: 10.18721/JCSTCS.13102

12. Молодяков С.А. Фотоприемники в системах потоковой обработки сигналов и изображений. СПб. : Изд-во Политехн. ун-та, 2014 .- 134 с.
13. Никитин В. В., Цыцулин А. К. Телевидение в системах физической защиты: Учеб. пособие/ СПб: СПбГЭТУ «ЛЭТИ», 2001. – 132 с.
14. Лавров А.П. Молодяков С.А. Оптоэлектронный процессор для регистрации радиоизлучения пульсаров. // Приборы и техника эксперимента.- 2015.- №1.- С.136–145.
15. Игнатов А.С., Круг П.Г. Интеллектуальные видеокамеры систем регулирования автомобильных потоков в черте крупных городов. // Промышленные АСУ и контроллеры. 2011. № 8. С. 20-23.
16. Лавров А.П., Молодяков С.А. Возможности построения процессоров обработки сигналов в виде гибридных оптоэлектронных микросхем // Научно-технические ведомости СПбГТУ Информатика. Телекоммуникации. Управление, 2008.- №5.- Т.65.- С. 144-151.
17. Молодяков С.А., Иванов С.И. Оптоэлектронный процессор в многоканальном радиометре // Информационно-управляющие системы.- 2009.- N2.- С. 10-16.
18. Лавров А.П., Молодяков С.А., Саенко И.И. Акустооптические процессоры в радиоастрономических приемниках // Антенны.- 2009.- №7.- С.45-55.
19. Есепкина Н.А., Молодяков С.А., Саенко И.И. Организация синхронного накопления на матричном ПЗС-фотоприемнике в модуляционном спектрометре. // Письма в ЖТФ.- 1986.- Т.12.- №2.- С.118-123.
20. Holst G.C., Lomheim T.S. CMOS/CCD Sensors and Camera Systems. SPIE Press, 2007.- 376p.
21. Chong-Min Kyung. Theory and Applications of Smart Cameras. Springer, 2016.- 366p
22. Молодяков С.А. Применение ПЗС-фотоприемников для предварительной

обработки сигналов //Датчики и системы. 2014.- № 5 (180).- С. 2-10.
23. Березин В. В. Твердотельная революция в телевидении: Телевизионные системы на основе приборов с зарядовой связью, систем на кристалле и видеосистем на кристалле/ В. В. Березин, А. А. Умбиталиев, Ш. С. Фахми, А. К. Цыцулин, Н. Н.Шипилов; М.: Радио и связь, 2006. – 312с.
24. Молодяков С.А. Системное проектирование оптоэлектронных процессоров обработки сигналов. СПб.: Изд-во Политехн. ун-та, 2011.- 226 с.
25. Сетевые камеры http://www.axis.com/global/ru/products/network-cameras
26. Сан-Мигель Х., Микелони К., Шуп К., Джан-Лука Форести, Кавальяро А. Умная сеть интеллектуальных камер //«Открытые системы», 2014.- № 06.- С.32-35.
27. Левко Г.В. Крупноформатные ПЗС И ПЗС мозаики //Вопросы радиоэлектроники, сер. Техника телевидения.- 2013.- вып. 1
28. Молодяков С.А. Специализированная видеокамера режима временной задержки и накопления для обработки сигналов. // Вопросы радиоэлектроники. Серия: Техника телевидения.- 2017.- № 1.- С. 31-38.
29. Оптоэлектронные процессоры с ПЗС-фотоприемнтками. Конвейерная обработка сигналов // Информационно-управляющие системы.- 2008.- N6.- С.2-8.
30. Лавров А.П., Молодяков С.А., Саенко И.И. Акустооптоэлектронные устройства в радиоастрономических приемных комплексах // Научно-технические ведомости СПбГПУ. Серия: Информатика. Телекоммуникации. Управление, 2010.- №4(103).- С.233-242
31. Молодяков С.А. Особенности и алгоритмы цифровой обработки сигналов в оптоэлектронных процессорах // Цифровая обработка сигналов. 2013. №3. С. 61-66.
32. Молодяков С.А. Управление информационными характеристиками фотоприемника на приборе с зарядовой связью в устройстве ввода изображения в ЭВМ.// Приборы и техника эксперимента.- 1987.- №3.- С.71-

75.
33. Молодяков С.А. Методика использования в цифровых камерах пульсарных процессоров кадровых ПЗС-фотоприемников в режиме временной задержки и накопления // Наука и образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2013.- № 5.- С.163-182 DOI: http://dx.doi.org/10.7463/0513.0577481
34. Васильев Алексей Программирование на Python в примерах и задачах Издательство Бомбора (Эксмо), 2020. - 616 с.
35. Лутц М. Программирование на Python. Издательство Символ, 2020. - 992 с.
36. Солем Ян Эрик Программирование компьютерного зрения на Python. ДМК-Пресс, 2016 312 с.
37. Шолле Франсуа Глубокое обучение на Python. СПб.: Питер, 2018. - 400с.
38. Саввин С.В., Сирота А.А. Методы суперпиксельной сегментации и их применение для анализа изображения с разнородной текстурой. Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. 2016. № 4. С. 165-173.
39. Gavrilova N. M., Dailid I. A., Molodyakov S. A., Boltenkova E. O., Korolev I. N., Popov P. A. Application of computer vision algorithms in the problem of coupling of the locomotive with railcars // 2018 International Symposium on Consumer Technologies (ISCT). P. 1 – 4
40. Deylid I., Molodyakov S., Tyutin B. Development of an Algorithm for Determining the Railway Tracks on Video Image. In: Voinov N., Schreck T., Khan S. (eds) Proceedings of International Scientific Conference on Telecommunications, Computing and Control. Smart Innovation, Systems and Technologies, 2021, vol 220. Springer, Singapore. P.27-35
41. Молодяков С.А. Преподаватель в вузе: из опыта повседневной жизни // Высшее образование в России. 2016. № 3 (199). С. 91–98.
42. Молодяков С. А. Компьютерное зрение: лабораторный практикум ; Санкт-Петербургский политехнический университет Петра Великого. — Санкт-Петербург, 2020 173с.

Санкт-Петербургский политехнический университет

Из приветствия ректора Руцкого А.И. «Начиная с первых лет существования, Санкт-Петербургский политехнический университет готовил высококвалифицированных инженеров для промышленности. И сегодня это направление остается главным. Качественно изменилось содержание подготовки. Мир сегодня стремительно меняется. В России назрела острая необходимость формирования новой экономики – экономики знаний, лидерства и инноваций, в основе которой лежит интеграция образования, науки и промышленности… Одна из наших главных задач – подготовка специалистов мирового уровня, способных работать на передовых производственных линиях, сочетая исследовательскую, проектную и предпринимательскую деятельность.»

В СПбПУ можно получить образование по самым разным направлениям – от машиностроения и телекоммуникаций до материаловедения и кибербезопасности. В нашей политехнической семье более 30 тысяч студентов

В 2020 году Политех стал первым среди российских вузов в рейтинге THE University Impact Rankings и получил статус научного центра мирового уровня «Передовые цифровые технологии».

Высшая школа программной инженерии (ВШПИ) СПбПУ

ВШПИ осуществляет подготовку бакалавров, магистров и аспирантов.

Подготовка бакалавров по направлениям:

09.03.04 «Программная инженерия»

02.03.02 «Фундаментальная информатика и информационные технологии»

Подготовка магистров по направлениям:

09.04.04 «Программная инженерия»

02.04.02 «Фундаментальная информатика и информационные технологии»

Магистерские программы:

- 09.04.04_01 Технология разработки и сопровождения качественного программного продукта
- 09.04.04_02 Основы анализа и разработки приложений с большими объемами распределенных данных
- 09.04.04.04 ИТ-инфраструктура предприятия
- 02.04.02_02 Проектирование сложных информационных систем

Подготовка аспирантов по направлению:

09.06.01 «Информатика и вычислительная техника»

Специальность 05.13.11 «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»

Программная инженерия— это область компьютерной науки и технологии, которая занимается построением программных систем… Это инженерная дисциплина, которая связана со всеми аспектами производства программного обеспечения от начальных стадий создания спецификации до поддержки

системы после сдачи в эксплуатацию. Термин – software engineering (программная инженерия) — впервые был озвучен в октябре 1968 года на конференции подкомитета НАТО по науке и технике (г.Гармиш, Германия).

Суть методологии программной инженерии состоит в применении систематизированного, научного и предсказуемого процесса проектирования, разработки и сопровождения программных средств.

В ВШПИ работают совместные научно-образовательные центры и кооперации:

«Политехник—Dell Technologies»

Основные направления исследований и разработок, развиваемых в Центре:

- Наука о данных и аналитика больших объемов информации
- Облачные инфраструктуры, вычисления и сервисы
- Управление информацией и распределенным хранением данных
- Архитектура резервного копирования и восстановления данных

«**Политехник—EPAM»**

EPAM предоставляет возможность будущим ИТ-специалистам получать практические навыки и знания в области разработки и тестирования программного обеспечения промышленного уровня параллельно с основным обучением. Для студентов СПбПУ есть возможность пройти бесплатное обучение в учебном центре EPAM.

«Политехник—FirstLine»

Центр «Политехник—FirstLine» организован в 2016 году. Центр создан с целью подготовки специалистов для работы в международной IT-индустрии, а также для проведения совместных научных исследований в области анализа и обработки больших данных в банковских и налоговых системах

«Политехник-Geoscan»

Группа Геоскан впервые в мире решила задачу в одной из областей искусственного интеллекта – автоматическое создание 3D-модели объекта по серии разноракурсных фотографий.

«Политехник-Luxoft»

Компания проводится обучение ИТ-специалистов по ключевым направлениям разработки программного обеспечения. Курсы от экспертов-практиков: системный и бизнес-анализ, архитектура ПО, ручное и автоматизированное тестирование ПО, Big Data и машинное обучение, управление проектами и Agile.

Из истории ВШПИ:

В апреле 1961 года в космос полетел Гагарин. Самое непосредственное отношение к полету Гагарина имел Ленинградский политехнический институт (ЛПИ) им. М. И. Калинина (ныне С. Петербургский политехнический университет Петра Великого — СПбПУ Петра Великого) и кафедра «Информационные и управляющие системы» (ныне Высшая школа программной инженерии), где проводились важные работы по

информационному обеспечению полета Гагарина. Не менее, а может быть и более важное место в работах кафедры и созданного при ней ОКБ ЛПИ занимали проблемы обороноспособности страны. Заведовал кафедрой в то время Тарас Николаевич Соколов. Важность работ, проводимых в ЛПИ, подтверждается, в частности, текстом выступления Главнокомандующего РВСН (ракетные войска стратегического назначения) генерал-полковника И. Д. Сергеева 9 февраля 1994 г. на конференции «История строительства и развития Ракетных войск стратегического назначения»:

«В этот технический прорыв, определивший дальнейшее развитие Ракетных войск стратегического назначения, внесли неоценимый вклад конструкторские бюро, возглавляемые Владимиром Николаевичем Челомеем, Михаилом Кузьмичем Янгелем, Сергеем Павловичем Королевым, Николаем Алексеевичем Пилюгиным, Тарасом Николаевичем Соколовым.»

Профессор ЛПИ Т. Н. Соколов почетно входит в блистательный список из четырех академиков – Генеральных конструкторов межконтинентальных ракет (Челомей, Янгель, Королев) и Генерального конструктора бортовых систем управления (Пилюгин), а коллектив ОКБ ЛПИ стоит рядом с коллективами прославленных КБ и НИИ.

Роль Т. Н. Соколова в проводимых в ЛПИ работах подтверждается и следующим текстом из «Хроники основных событий истории РВСН», написанной под общей редакцией Главнокомандующего РВСН генерал-полковника И. Д. Сергеева в 1994 г.:

«СОКОЛОВ Тарас Николаевич (17.04.1911 – 15.09.1979). Главный конструктор НПО «Импульс». Доктор технических наук (1951). Профессор (1953). Герой Социалистического Труда (1970). Лауреат Ленинской премии (1959) и двух Государственных премий (1949, 1977).

Один из основных создателей систем управления. Внес значительный вклад в работы в области исследования космического пространства, запуск первого ИСЗ и первого полета человека в космос. Является родоначальником автоматизации процессов управления войсками и стратегическим оружием в РВСН. Под его руководством и с его непосредственным участием созданы и внедрены в войска высоконадежные АСУ и системы дистанционного управления оружием. Разработанные им технические решения и сегодня являются базовыми в части создания перспективных АСУ РВСН и СЯС в целом».

(Здесь: ИСЗ – искусственный спутник Земли, АСУ – автоматизированная система управления, СЯС – стратегические ядерные силы.)

Подробная информация о Высшей школе программной инженерии имеется на сайте:

https://hsse.spbstu.ru/