

Э. Д. Шакирьянов

КОМПЬЮТЕРНОЕ ЗРЕНИЕ НА PYTHON®

ПЕРВЫЕ ШАГИ



Э. Д. Шакирьянов

КОМПЬЮТЕРНОЕ ЗРЕНИЕ НА PYTHON®

ПЕРВЫЕ ШАГИ

Электронное издание



Москва
Лаборатория знаний
2021

УДК 087.5 + 004.42 + 004.896

ББК 32.81 + 32.973

Ш17

Серия основана в 2020 г.

Шакирьянов Э. Д.

Ш17 Компьютерное зрение на Python®. Первые шаги / Э. Д. Шакирьянов. — Электрон. изд. — М. : Лаборатория знаний, 2021. — 163 с. — (Школа юного инженера). — Систем. требования: Adobe Reader XI ; экран 10". — Загл. с титул. экрана. — Текст : электронный.

ISBN 978-5-00101-944-2

В книге изложен учебный курс для школьников, начинающих изучать компьютерное зрение с языком программирования Python и библиотекой OpenCV. Описаны особенности установки языка Python, различных библиотек, в том числе OpenCV, и операционной системы Raspbian. Материал разделен на три отдельные темы: программирование на Python, поиск и выделение цветных объектов на графическом изображении и в видеопотоке средствами OpenCV, программирование колесной робоплатформы под управлением Raspberry Pi 3, оснащенной CSI-камерой.

Большую помощь читателю окажут многочисленные иллюстрации и листинги программных кодов, а также ссылки на источники и интернет-ресурсы.

Книга будет полезна школьникам среднего и старшего возраста, педагогам дополнительного образования и всем начинающим изучать компьютерное зрение с помощью языка программирования Python и открытой библиотеки компьютерного зрения OpenCV-Python.

УДК 087.5 + 004.42 + 004.896

ББК 32.81 + 32.973

Деривативное издание на основе печатного аналога: Компьютерное зрение на Python®. Первые шаги / Э. Д. Шакирьянов. — М. : Лаборатория знаний, 2021. — 160 с. : ил. — (Школа юного инженера). — ISBN 978-5-00101-318-1.

6+

В соответствии со ст. 1299 и 1301 ГК РФ при устранении ограничений, установленных техническими средствами защиты авторских прав, правообладатель вправе требовать от нарушителя возмещения убытков или выплаты компенсации

ISBN 978-5-00101-944-2

© Лаборатория знаний, 2021

Оглавление

Предисловие	5
Глава 1. Начало программирования на Python	7
О языке Python	7
Установка Python и дополнительных библиотек	8
Синтаксис языка Python	12
Программирование на языке Python с помощью исполнителя Turtle	15
Turtle-методы	15
Изучение функций с помощью графического исполнителя Turtle	19
Методы TurtleScreen	23
Итоги главы 1	26
Глоссарий к главе 1	27
Список литературы к главе 1	29
Глава 2. Начало работы с OpenCV-Python	30
Установка библиотеки OpenCV-Python	30
Основные операции в OpenCV-Python	35
Открытие файла с изображением	37
Изменение размера изображения	39
Вырезка фрагмента изображения	41
Поворот изображения	44
Зеркальное отражение изображения по осям	46
Сохранение изображения в файл на локальный диск	48
Поиск объекта по цвету	49
Цветовые пространства	52
Поиск объекта в цветовом пространстве HSV	54
Определение координат найденного объекта	57
Отображение видео в OpenCV-Python	60
Отображение видеоданных с web-камеры	60
Отображение видеоданных из файла	63
Поиск цветных объектов на видео с помощью OpenCV-Python	65
Суммирование цветowych масок	68

Настройка цветового фильтра средствами OpenCV.....	73
Поиск цветных объектов на видео в реальном времени.....	79
Выделение контуров объектов с помощью OpenCV-Python	84
Выделение прямоугольных и эллиптических контуров.....	90
Выделение прямоугольных контуров.....	90
Фильтрация прямоугольных контуров.....	94
Вычисление угла поворота прямоугольного контура.....	99
Выделение и фильтрация эллиптических контуров	105
Выделение контуров дорожных знаков в видеопотоке.....	110
Итоги главы 2.....	115
Глоссарий к главе 2.....	116
Список литературы к главе 2.....	118
Гл в 3. Программирование Raspberry Pi.....	119
Микрокомпьютеры Raspberry Pi.....	119
Установка операционной системы Raspbian	121
Установка OpenCV на Raspberry Pi 3.....	125
Удаленное управление Raspberry Pi 3	131
Подключение модулем камеры CSI для Raspberry Pi и захват видео	136
Управление моторами с помощью Raspberry Pi на примере робота AlphaBot.....	142
Программирование движения робота AlphaBot вдоль черной линии	149
Итоги главы 3.....	155
Глоссарий к главе 3.....	156
Список литературы к главе 3.....	157
3 ключение.....	159

Предисловие

Ни для кого не секрет, что роботы все чаще встречаются в нашей повседневной жизни. Роботы развлекают детей, летают в космос, помогают спасателям, управляют автомобилями и боевой техникой. С каждым поколением они становятся все совершеннее. О новых разработках в робототехнике мы узнаем с экранов телевизоров и на сайтах в Интернете, со страниц журналов и книг. Современные роботы способны видеть, распознавать объекты окружающего их мира и выполнять сложные операции. Более того, современные роботы способны обучаться, адаптируясь к внешней среде. Поведение таких роботов очень близко к поведению живых существ.

Значимой составляющей в подобных роботах является система компьютерного зрения. Эта технология позволяет роботам с помощью специальных алгоритмов и программ анализировать поток данных, который они получают с видеокамеры. Главными задачами компьютерного зрения является обнаружение, идентификация и классификация объектов окружающего мира, которые попали в поле зрения видеокамеры. Развитые системы компьютерного зрения могут распознавать различные активности, которые проявляют обнаруженные объекты.

До недавнего времени изучение, проектирование и создание устройств, оснащенных простейшим компьютерным зрением, требовали бы от разработчика больших интеллектуальных и материальных ресурсов. Но прогресс не стоит на месте. С появлением недорогих компактных микрокомпьютеров, а также доступного программного обеспечения изучение и программирование несложных систем с компьютерным зрением стало возможно даже для школьников.

Книга, которую вы держите в руках, ориентирована на читателей, начинающих изучать компьютерное зрение на языке Python. Весь материал разбит на три главы и снабжен иллюстрациями, ссылками на источники и программными кодами. После каждой главы приводится список литературы и интернет-ресурсов, которые могут быть полезны для дополнительного изучения.

Первая глава представляет собой краткое введение в программирование на языке Python. В настоящее время доступно множество различных источников и интернет-ресурсов, с помощью которых можно освоить основы программирования на Python, поэтому здесь этот вопрос не рассматривается подробно. Начальные навыки программирования в Python довольно легко освоить самостоятельно. Синтаксис языка Python прост и понятен, а его особенности можно постичь в процессе практики.

Вторая глава посвящена ключевой теме книги — изучению основ компьютерного зрения. Одним из самых мощных и доступных инструментов программирования компьютерного зрения на Python является библиотека **OpenCV (Open Source Computer Vision Library)** — библиотека компьютерного зрения с открытым исходным кодом. Наиболее важные, с точки зрения автора, вопросы, первоначально необходимые для работы с OpenCV, изложены в этой главе. Весь материал главы носит прикладной характер и условно разбит на пять тем:

- 1) установка OpenCV;
- 2) основные операции в OpenCV;
- 3) выделение объекта по цветовой маске;
- 4) работа с видеопотоком;
- 5) контурный анализ.

В третьей главе рассмотрены вопросы, связанные с подготовкой микрокомпьютера Raspberry Pi 3 к работе, установкой на него библиотеки OpenCV и программированием движения робоплатформы, оснащенной видеокамерой, вдоль черной линии. На сегодняшний день для реализации систем с компьютерным зрением самой подходящей по многим критериям является, пожалуй, техническая платформа на основе семейства **микрокомпьютеров Raspberry Pi**. Вычислительных ресурсов Raspberry Pi хватает для большинства задач начального уровня.

Предлагаемая книга не является исчерпывающим пособием по изучению компьютерного зрения с помощью OpenCV-Python. Здесь в сжатой форме собран материал, который будет полезным для начинающих изучать язык программирования Python и основы компьютерного зрения с библиотекой OpenCV-Python.

Итак, давайте окунемся в интересный мир программирования и робототехники! Вместе разберемся с принципами, на которых реализовано компьютерное зрение роботов. Настроим свой первый микрокомпьютер и научим его управлять робоплатформой по картинке с видеокамеры.

Начало программирования на Python

О языке Python

Python — это интерпретируемый язык программирования. Исходный код преобразуется в машинный частью по мере его выполнения специальной программой, которая называется **интерпретатор**. Язык Python придумал голландец Гвидо ван Россум примерно в 1991 г.

После того как Россум разработал язык Python и выложил его в Интернет, появилось целое сообщество программистов, которые активно его улучшают и совершенствуют. Новые версии языка регулярно выходят на его официальном сайте¹ [1].

Python отличается своим уникальным синтаксисом. Даже начинающий программист может легко научиться кодировать на этом языке программирования. Одной из особенностей языка является его требовательность к отступам, заставляющая программистов воспитывать в себе культуру написания программных кодов. В результате программный текст получается структурированным, легко читается и понимается. Добавим еще, что Python — это полноценный универсальный объектно-ориентированный язык программирования.

ПОЯСНЕНИЕ

Технология объектно-ориентированного программирования² основана на представлении программы в виде взаимодействия не-

¹ Python Software Foundation. // Официальный сайт разработчика Python. URL: <http://python.org>

² Объектно-ориентированное программирование. // Викиучебник. URL: https://ru.wikibooks.org/wiki/Объектно-ориентированное_программирование

которого числа **объектов**. При этом каждый **объект** принадлежит к конкретному **классу** и обладает характерным для этого класса набором **свойств** и **методов**. Например, Шарик и Бобик являются объектами, которые взаимодействуют между собой (лают), при этом оба относятся к классу «Собаки». Следовательно, они, как и все собаки, имеют четыре лапы, хвост и умеют лаять. А Барсик является объектом, который принадлежит к классу «Кошки». Он тоже имеет четыре лапы и хвост, но в отличие от собак умеет мяукать. И в то же время оба класса «Собаки» и «Кошки» имеют общие свойства: четыре лапы и хвост, которые были унаследованы от более широкого класса «Домашние животные». К объектно-ориентированным языкам относятся языки C/C++, Java, Object Pascal.

Кроме того, Python является свободно распространяемым языком на основании лицензии, совместимой с GNU General Public License.

Кроссплатформенные возможности Python позволяют программировать на этом языке под различными операционными системами, например Windows, Linux, Mac OS и др. Это добавляет привлекательности Python среди программистов.

Установка Python и дополнительных библиотек

Установка Python не представляет особых трудностей. Предварительно необходимо скачать установщик Python с официального сайта [1]. В целях безопасности ваших данных на компьютере не рекомендуется скачивать Python из сторонних ресурсов. Итак, заходим на сайт Python и в меню *Downloads* скачиваем последнюю актуальную версию для своей операционной системы (**рис. 1.1**). В нашем случае это будет операционная система Windows.

После окончания загрузки следует открыть папку (по умолчанию установочный файл сохранится в папке *Загрузки*) с загруженным исполняемым файлом и запустить его. В диалоговом окне установщика (**рис. 1.2**) следует поставить галочку напротив пункта *Add Python 3.7 to PATH*. Затем ждем завершения процесса установки. По окончании установки Python запускается из меню *Пуск*. Более подробные инструкции по установке, в том числе и на другие операционные системы, можно посмотреть здесь¹ [2, 3].

¹ Python. Урок 1. Установка. // Devpractice. Разработка программного обеспечения, технологии и наука. URL: <https://devpractice.ru/python-lesson-1-install/>

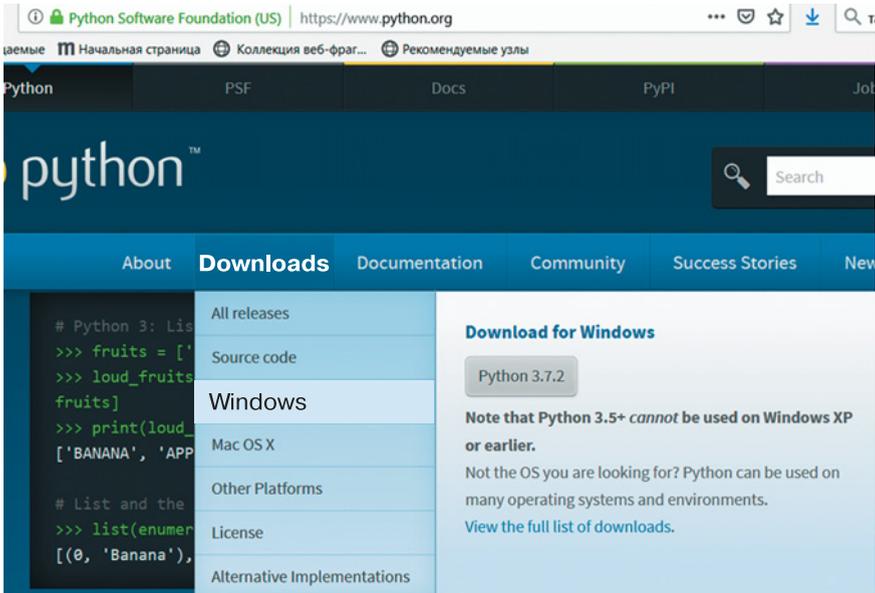


Рис. 1.1. Меню загрузки Python для операционной системы Windows

Для работы в Python, кроме стандартных библиотек и пакетов, иногда требуется установить дополнительные специализированные библиотеки. В настоящее время под различные задачи разработано огромное количество библиотек и пакетов для Python. Чтобы установить дополнительные пакеты и библиотеки, используют утилиту **pip**.

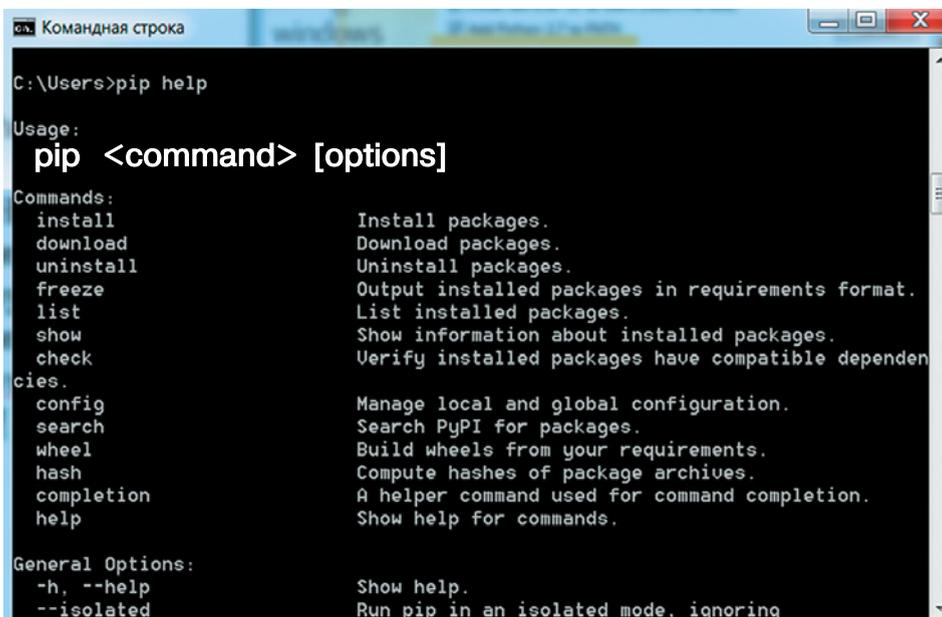


Рис. 1.2. Диалоговое окно установщика с выбором параметров установки языка Python

ПОЯСНЕНИЕ

pip — система управления пакетами, которая используется для установки и управления программными пакетами и библиотеками, написанными на Python. Начиная с версии Python 2.7.9 и Python 3.4, дистрибутив Python по умолчанию содержит утилиту управления пакетами **pip** (или **pip3** для Python 3).

С помощью **pip**, используя соединение с Интернетом, можно легко установить практически любой дополнительный пакет или библиотеку. Работа с **pip** осуществляется в командной строке Windows (рис. 1.3).



```
Командная строка
C:\Users>pip help

Usage:
pip <command> [options]

Commands:
  install           Install packages.
  download          Download packages.
  uninstall         Uninstall packages.
  freeze           Output installed packages in requirements format.
  list             List installed packages.
  show             Show information about installed packages.
  check            Verify installed packages have compatible dependencies.
  config           Manage local and global configuration.
  search          Search PyPI for packages.
  wheel           Build wheels from your requirements.
  hash            Compute hashes of package archives.
  completion      A helper command used for command completion.
  help            Show help for commands.

General Options:
  -h, --help      Show help.
  --isolated      Run pip in an isolated mode, ignoring
```

Рис. 1.3. Командная строка Windows

Установка нового пакета производится следующим образом. Например, чтобы установить пакет **numpy**, необходимо ввести в командной строке:

```
pip install numpy
```

или для Python 3:

```
pip3 install numpy
```

В Linux установка этого же пакета (подробнее об этом будет в главе 3) выполняется с помощью команды от имени администратора:

```
sudo pip3 install numpy
```

ПОЯСНЕНИЕ

Numpy — это пакет языка Python, предназначенный для работы с большими многомерными массивами и матрицами. Содержит в себе большую библиотеку высокоуровневых (и очень быстрых) математических функций для операций с этими массивами^{1, 2}. В дальнейшем этот пакет нам понадобится для работы с компьютерным зрением. Данный пакет является обязательным компонентом для работы с OpenCV.

Утилита **pip** выполнит загрузку пакета и распакует его в специальную папку с установленным Python. Как видите, установка дополнительного пакета с помощью **pip** не представляет большого труда. Полная документация по работе с **pip** представлена на сайте³ [4]. Ниже приведены основные команды **pip**, где `<package_name>` — название устанавливаемого вами пакета:

```
pip help — помощь по доступным командам;
pip install <package_name> — установка пакета(ов);
pip uninstall <package_name> — удаление пакета(ов);
pip list — список установленных пакетов;
pip show <package_name> — показывает информацию
об установленном пакете;
pip search — поиск пакетов по имени;
pip install-U — обновление пакета(ов);
pip install--force-reinstall — при обновлении переустановить пакет, даже если он последней версии.
```

¹ Документация по numpy. URL: <https://docs.scipy.org/doc/numpy/reference/>

² Numpy для начинающих. URL: <https://pythonworld.ru/numpy>

³ pip — The Python Package Installer. URL: <https://pip.pypa.io/en/stable/>

Более подробно с установкой и удалением дополнительных пакетов, в том числе и без использования утилиты **pip**, можно ознакомиться в [5–7].

Программирование в Python предусматривает два режима: консольный и интерактивный. В первом случае в окне будет высвечено приглашение к вводу команды (`>>>`). В этом режиме интерпретатор исполняет введенную команду сразу после нажатия клавиши **enter**. Другой режим — это работа во встроенной среде разработки IDLE, в которой тоже есть интерактивный режим работы. Но в отличие от консольного варианта здесь можно писать полноценные программы и наблюдать подсветку синтаксиса (в зависимости от значения синтаксической единицы она выделяется определенным цветом¹). Программы на Python часто называют **скриптами**, которые с помощью среды IDLE можно сохранять в файлы с расширением **.py**.

Чтобы создать скрипт в среде IDLE, после запуска программы в меню следует выбрать команду *File* → *New Window* (Ctrl + N). В результате откроется новое окно, в котором пишется программный код. И наконец, чтобы его запустить для исполнения, необходимо выполнить команду меню *Run* → *Run Module* (F5). После этого в консольном окне Python появится результат выполнения кода (**рис. 1.4**).

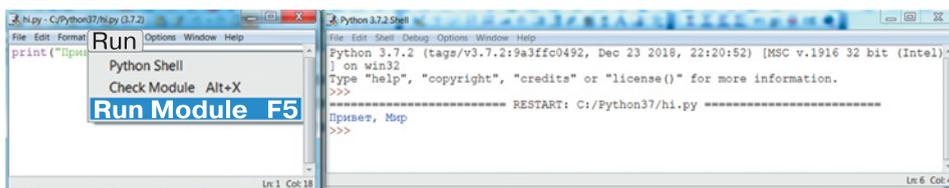


Рис. 1.4. Окно программного кода (слева) и консольное окно (справа)

Синтаксис языка Python

Программирование на Python с самого начала приучает начинающего программиста писать правильный и удобочитаемый код (соблюдать принципы, называемые общим словом **readability**). В отличие от других языков в Python значение имеют не только лексемы языка.

¹ Если набирать код, не сохранив сначала файл, то подсветка синтаксиса будет отсутствовать.

ПОЯСНЕНИЕ

Лексема — это структурная единица языка программирования, которая состоит только из разрешенных символов алфавита языка и не содержит в своем составе другие структурные единицы языка или символы. Например, лексемами языка программирования являются: ключевые слова языка, имена переменных, имена констант, имена функций, знаки операций и т. д. Все ключевые слова (иначе — зарезервированные или служебные) являются постоянной частью языка программирования и записываются в точности так, как это установлено правилами языка. Ключевые слова нельзя использовать в качестве имен других структурных единиц языка. В Python строчные и ПРОПИСНЫЕ буквы **различаются**.

В Python особое внимание уделяется **отступам** строк в программном коде. Наличие или отсутствие отступов строк определяет правильность программного кода с точки зрения логики и синтаксиса. Другими словами, если отступы выставлены неверно, то может нарушиться логика работы программы или возникнуть синтаксическая ошибка в, казалось бы, «правильном» программном коде.

Приведем несколько **правил**, которые нам понадобятся на первом этапе и которые надо соблюдать, чтобы избежать ошибок в программном коде.

- Конец строки является концом инструкции — команды или блока, в конце строки точка с запятой не ставится.
- В Python вложенные инструкции оформляются единообразно. Сначала пишется основная инструкция, завершающаяся двоеточием. Затем на следующей строке размещается вложенный блок команд. Он может состоять также из нескольких строк, но все они будут начинаться с одинакового отступа — на «шаг» вправо относительно основной инструкции.

Основная инструкция:

Вложенный блок инструкций

- Инструкции, относящиеся к одному блоку, всегда имеют одинаковый отступ! «Шаг» обычно равен четырем пробелам или знаку табуляции Tab¹. Чем глубже степень вложения, тем большее количество пробелов требуется, однако всегда оно кратно четырем.

¹ Среда автоматически заменяет нажатие клавиши Tab на 4 пробела. Это сделано для удобства пользователя, однако не приветствуется создателями языка.

Основная инструкция:

```
Вложенная инструкция 1
    Вложенная инструкция 1.1
    Вложенная инструкция 1.2
Вложенная инструкция 2
    Вложенная инструкция 2.1
    Вложенная инструкция 2.2
```

- В некоторых случаях, например для удобочитаемости программного кода, несколько инструкций можно записать в одной строке через точку с запятой¹:

```
p = 3; q = 5; s = input("Введите число:")
```

- Длинную инструкцию разрешается записывать в несколько строк, поместив ее внутри круглых, квадратных или фигурных скобок:

```
while (i < (a + b)/2 or
       q = False):
    i +=1
```

- Вложенная инструкция может располагаться в той же строке, что и тело основной, если тело вложенной инструкции не является составным, то есть состоит только из одной команды, например:

```
for i in range(1, n): f = f * i
```

Следуя этим правилам, на первом этапе можно в большинстве случаев исключить ошибки синтаксиса при написании кода. Конечно, полное понимание, как правильно писать код на Python, невозможно без практических навыков.

ДОПОЛНЕНИЕ

Более полную информацию, как правильно писать программы на языке Python, можно получить в официальном руководстве по написанию кода. Кроме того, в соглашении будет полезно ознакомиться с тем, что такое документирование кода и для чего это необходимо.

PEP 8 — Style Guide for Python Code // Python Software Foundation.
URL: <https://www.python.org/dev/peps/pep-0008>

¹ Не увлекайтесь этим чрезмерно, чтобы не получить обратный эффект.

PEP 8 — руководство по написанию кода на Python // Python 3 для начинающих. URL: <https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html>

PEP 257 — Docstring Conventions // Python Software Foundation. URL: <https://www.python.org/dev/peps/pep-0257>

Документирование кода в Python. PEP 257 // Python 3 для начинающих. URL: <https://pythonworld.ru/osnovy/dokumentirovanie-koda-v-python-pep-257.html>

Программирование на языке Python с помощью исполнителя Turtle

Библиотека **Turtle** («черепаха») представляет собой удобный исполнитель, средой которого является двумерное графическое поле. Эта библиотека очень хороша для тех, кто только начинает изучение основ алгоритмизации и программирования. Она уже встроена в Python, поэтому ее не надо отдельно устанавливать. Библиотека **Turtle** содержит набор простых и понятных команд, результат выполнения которых наглядно отображается в графическом окне. Также с помощью **Turtle** можно легко перестроиться на программирование в Python. Ниже мы разберем несколько примеров программ, экспериментируя с которыми можно самостоятельно развить в себе навыки программирования в Python. Рассмотрим работу некоторых из множества методов **Turtle**.

Turtle-методы

Для рисования и перемещения по рабочему полю в исполнитель **Turtle** заложен ряд простых для понимания методов. Главный инструмент **Turtle** — **перо**. Оно имеет два состояния: опущено и поднято (по умолчанию опущено). Кроме того, можно задать толщину и цвет пера. Рисование осуществляется путем перемещения исполнителя **Turtle** с опущенным пером. Ниже дан список команд, с помощью которых можно запрограммировать **Turtle** для рисования простейших рисунков. Символ «|» означает выбор одного из нескольких вариантов вызова данной команды.

Методы перемещения и рисования

`forward(n)` | `fd(n)` — перемещение исполнителя вперед на n пикселей;
`backward(n)` | `bk(n)` | `back(n)` — перемещение исполнителя назад на n пикселей;
`right(a)` | `rt(a)` — поворот исполнителя вправо (по часовой стрелке) на a градусов;
`left(a)` | `lt(a)` — поворот влево (против часовой стрелки) на a градусов
`goto(x, y)` | `setpos(x, y)` | `setposition(x, y)` — перейти в точку с координатами (x, y) ;
`setx(x)` — сместиться в указанную координату x , сохраняя y ;
`sety(y)` — сместиться в указанную координату y , сохраняя x ;
`setheading(a)` | `seth(a)` — задать направление на a градусов;
`home()` — вернуться в точку с координатами $(0,0)$;
`circle(r)` — нарисовать окружность радиусом r ;
`dot(s, c)` — нарисовать точку в текущей позиции, где s — размер знака точки в пикселях; c — название цвета в кавычках, например «red», «yellow», «dimgray» и т. д.¹;
`stamp()` — создать клон «черепашки»;
`undo()` — отменить последнее действие;
`speed(s)` — задать скорость рисования s по шкале от 0 до 10 условных единиц.

Методы состояния

`pos()` — возвращает текущую позицию курсора (x, y) ;
`xcor()` — возвращает текущую позицию курсора по координате x ;
`ycor()` — возвращает текущую позицию курсора по координате y ;
`distance(x, y)` — возвращает значение расстояния от текущей позиции курсора до координаты точки (x, y) ;

Методы управления пером

`pd()` — опустить перо;
`pu()` — поднять перо;
`pensize()` | `width()` — установить толщину пера по шкале от 0 до 10 условных единиц;
`pencolor()` — устанавливает/сообщает цвет пера;

¹ Цветовая схема — названия цветов. URL: <https://undoshutdown.blogspot.com/2018/06/matplotlib-python.html>

`fillcolor()` — устанавливает/сообщает цвет заливки;
`begin_fill()` — начало заливки установленным цветом;
`end_fill()` — конец заливки установленным цветом.

Рассмотрим примеры двух простых программ для рисования:

- 1) прямоугольника с текущей позиции курсора;
- 2) окружности с центром в заданной точке.

1. Чтобы нарисовать прямоугольник, сделаем запрос двух целых чисел. Для хранения их значений создадим две **переменные**.

ПОЯСНЕНИЕ

Переменная — это именованная область памяти компьютера, выделенная для хранения данных. Доступ к данным и изменение их значения в ходе работы программы осуществляются с помощью *идентификатора* — уникального имени переменной. В Python для переменной не требуется специального объявления, она может быть объявлена в любом месте программы путем присваивания ей любого значения. По мере выполнения программы тип переменной может меняться. Более подробно о переменных и их типах в Python можно прочитать здесь^{1,2}. В основном мы будем использовать функции, одноименные с типами переменных, — функции преобразования типов данных. О том, что такое процедура или функция, мы подробно остановимся позднее. Приведем часть основных типов данных и процедур, к ним приводящих:

Тип данных	Описание	Приведение к типу (функция)	Пример
<code>int</code>	Целое число	<code>int()</code>	5 -3
<code>float</code>	Число с плавающей точкой	<code>float()</code>	5.0 -3.4
<code>str</code>	Строка как последовательный набор символов	<code>str()</code>	«Привет!»
<code>bool</code>	Логическая переменная	<code>bool()</code>	True False

¹ Переменные в Python. URL: <https://codelessons.ru/python/uroki-dlyanovichkov-python/peremennye-v-python.html>

² Типы данных в Python. URL: <http://pythonicway.com/python-data-types>

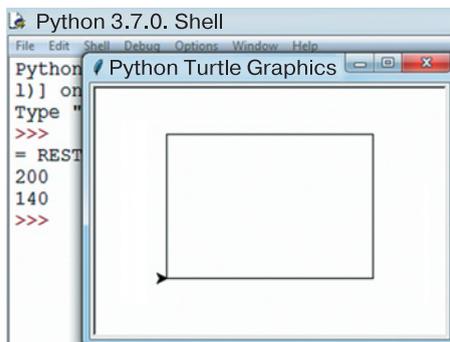


Рис. 1.5. Рисование прямоугольника с помощью исполнителя **Turtle**

Перейдем к практическому заданию. Пусть значения, записанные в переменные, будут определять длину и ширину некоторого прямоугольника. Дважды в цикле нарисуем одну за другой стороны прямоугольника: длину и ширину. Ниже представлен небольшой программный код. Результат работы программы изображен на **рис. 1.5**. После символа `#` даны комментарии, которые игнорируются исполнителем.

```
# подключение библиотеки Turtle
from turtle import * # загрузить всё из корневого
# каталога библиотеки turtle
# ввести с клавиатуры размеры прямоугольника (a * b)
a = int(input()) # ввод целого числа в переменную a
b = int(input()) # ввод целого числа в переменную b
# рисовать стороны прямоугольника
for i in range(0,2): # выполняем два раза
    fd(a) # вперед на a
    lt(90) # поворот налево на 90
    fd(b) # вперед на b
    lt(90) # поворот налево на 90
```

2. Стандартная функция **Turtle** для рисования окружности по умолчанию рисует ее из нижней точки, что порой не очень удобно. Составим небольшую программу, которая рисует окружность заданного радиуса с центром в заданной точке. Для этого надо поднять перо, сместиться в заранее рассчитанную точку, опустить перо и нарисовать окружность. При этом исполнитель **Turtle** должен быть ориентирован по умолчанию — слева направо. Результат работы программы представлен на **рис. 1.6**.

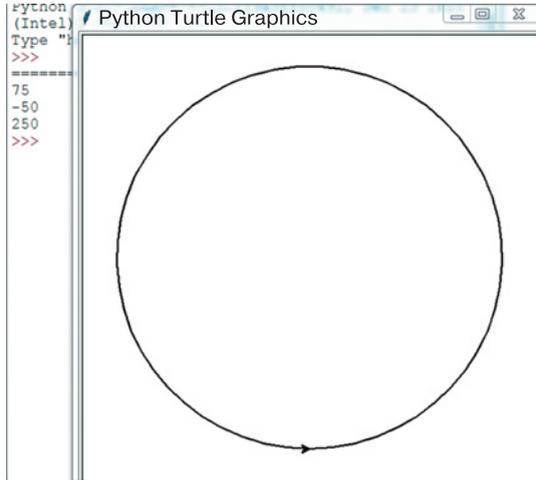


Рис. 1.6. Рисование окружности с центром в заданной точке с помощью исполнителя **Turtle**

```
# подключение библиотеки Turtle
from turtle import *
# ввод координат (x,y) центра окружности и радиуса r
x = int(input()) # запрос с клавиатуры числа x
y = int(input()) # запрос с клавиатуры числа y
r = int(input()) # запрос с клавиатуры числа r
# поднять перо
pu()
# сместиться в точку (x, y-r)
setx(x)
sety(y-r)
# опустить перо
pd()
# нарисовать окружность радиуса r
circle(r)
```

Изучение функций с помощью графического исполнителя Turtle

В практике программирования часто приходится писать программы с большим числом строк. Логика таких программ носит в определенной степени сложный характер, поэтому их удобно писать, разбивая

на отдельные, логически взаимосвязанные функциональные блоки. Реализацию таких блоков в программном коде можно осуществлять независимо, по мере отладки тестируя их на исполнение отдельных функций и на работу в связке. Для написания такого обособленного программного кода в Python используют **функции**.

ПОЯСНЕНИЕ

Процедуры и функции — это именованная часть программы, отвечающая за выполнение некоторой логически целостной подзадачи. По структуре в большинстве языков процедуры и функции условно состоят из двух частей: **заголовок**, где содержится имя процедуры или функции, и **тело**, где содержится их основной программный код. С понятиями процедура и функция связана такая методология, как **процедурное программирование**. Процедурное программирование предполагает разбиение сложной задачи на более простые, логически связанные подзадачи, реализация которых осуществляется через процедуры и функции. Применение процедур и функций позволяет создать программу с компактным и понятным кодом. Язык программирования может реализовывать парадигмы объектно-ориентированного и процедурного программирования одновременно.

Для определения функции в Python используется инструкция **def**. В общем виде функция записывается следующим образом:

```
def <имя_функции> (<список_аргументов>) :  
    <тело_функции>  
    [return <значение>]
```

В Python функция может принимать произвольное количество аргументов или не принимать их вовсе, а в качестве результата возвращать любые объекты. Если строка со служебным словом **return** опущена, то функция вернет результат **None**.

Рассмотрим примеры использования функций в программах с исполнителем **Turtle**. Мы уже научились изображать отдельно прямоугольники и окружности. Однако если потребуется изобразить несколько разных прямоугольников и окружностей, то нам придется написать объемный по количеству строк программный код. Используя функции, мы существенно упростим себе задачу. Для этого определим функции рисования прямоугольника с размерами $a \times b$ и окружности с центром в точке с координатами (x, y) и радиусом r .

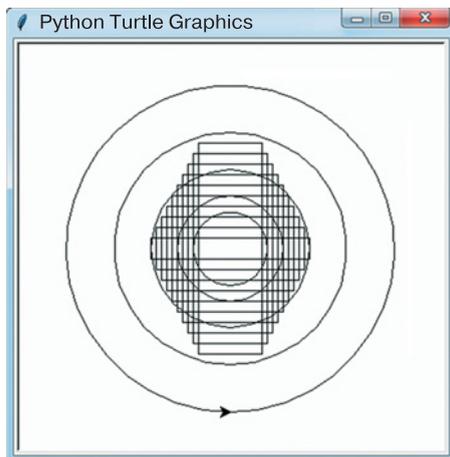


Рис. 1.7. Рисование исполнителем **Turtle** с помощью функций `rect()` и `ring()`

В качестве тела функций возьмем коды программ для рисования прямоугольника и окружности (см. рис. 1.5 и 1.6). При этом параметры, которые мы вводили с клавиатуры, теперь перейдут в определение функций соответственно. Результат работы программы изображен на **рис. 1.7**. При комментировании программы специально оставлены пустые строки — с этими командами мы уже знакомы.

```
from turtle import *  
# функция рисования прямоугольника (a x b)  
# определяем функцию с названием rect(a,b)  
def rect(a,b):  
    # тело функции rect(a,b)  
    for i in range(0,2):  
        # ???  
        fd(a)  
        # ???  
        lt(90)  
        fd(b)  
        lt(90)  
  
# функция рисования окружности с центром в (x,y)  
# с радиусом (r)  
def ring(x,y,r): # определяем функцию с названием  
ring(x,y,r)  
    # ???  
pu()
```

```

    # ???
    setx(x)
    # ???
    sety(y-r)
    # ???
    pd()
    # ???
    circle(r)
# Далее идет код основной программы
# пример использования функции rect(a,b) в цикле
n = 10
# запускаем цикл на 10 повторений
for i in range(0,n):
    # вызываем функцию rect
    rect(60+10*i,200-20*i)
    # ???
    pu()
    # даем команду сместиться в новую точку
    goto(-5*(i+1),10*(i+1))
    # ???
    pd()
# пример использования функции ring(x,y,r)
n = 5
# запускаем цикл на 5 повторений
for i in range(0,n):
    # вычисляем радиус окружности
    R = 30+5*(i+1)*(i+1)
    # вызываем функцию ring
    ring(30,100,R)

```

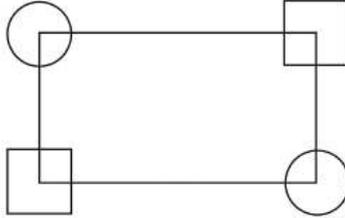
Попробуйте поэкспериментировать, меняя в формулах с переменной «i» некоторые числа. Запуская программу, посмотрите, как изменится картинка.

Чтобы окончательно закрепить понимание работы программы, подумайте и впишите комментарии вместо знаков «???».

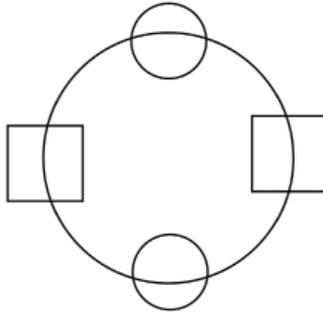
ЗАДАНИЯ

Используя функции для рисования прямоугольника и окружности с центром в заданной точке, напишите программы для выполнения следующих задач.

1. Ниже изображена картинка. Размеры прямоугольника: $a \times b$. Числа a и b запрашиваются с клавиатуры. Диаметр окружности $c \leq (1/2) a$, а сторона квадрата $c \leq (1/2) b$. Когда вы закончите составлять программу, напишите комментарии к ее строкам. *Подсказка.* При сравнении в Python замените \leq на $<=$.



2. Диаметр большой окружности равен a . Число a запрашивается с клавиатуры. Диаметр малых окружностей и сторона квадрата равны $c \leq (1/2) a$. Напишите программу и комментарии к ней.



Методы TurtleScreen

В исполнителе **Turtle** имеются методы управления окном и анимацией. Эта группа методов позволяет разнообразить функционал библиотеки **Turtle**.

Методы управления окном

`bgcolor()` — задает цвет фона окна, например `bgcolor («orange»)`;
`bgpic("<file_name.gif>")` — загружает фоновое изображение в формате *.gif;

`screensize(1200, 800, "green")` — задает размеры окна и цвет; если скобки пустые (), возвращает размер окна.

Методы управления анимацией

`delay(5)` — установка или возврат задержки отрисовки экрана в миллисекундах, приблизительно равной периоду последовательных обновлений холста;

`tracer(n, m)` — включает/выключает анимацию курсора и устанавливает задержку для обновления рисунков. Если задано `n` (может быть использовано для эффекта ускорения/замедления анимации), то выполняется только каждое `n`-е обновление экрана, если задано `m`, устанавливается задержка обновления.

Разберем небольшой пример, в котором демонстрируется применение собственных функций и простейшего элемента анимации — движущегося объекта на экране. Чтобы получить эффект движения какого-либо объекта, надо сделать следующее:

- 1) изобразить объект на экране;
- 2) сделать небольшую задержку для отображения содержимого экрана;
- 3) заменить цвет объекта цветом фона;
- 4) сделать небольшую задержку для отображения экрана;
- 5) рассчитать новые координаты местоположения объекта;
- 6) повторить действия, описанные в пунктах 1–5.

Почему это работает? Дело в том, что наше зрение обладает некоторой инертностью. Благодаря этому мы не замечаем мерцания люминесцентных ламп, смены кадров на видео и т. д. То же самое происходит и здесь. Мы успеваем увидеть объект, цвет которого отличается от цвета фона, но не успеваем увидеть, как он исчез и проявился вновь с небольшим смещением. Нам кажется, что объект просто движется. В результате получается анимационный эффект, который можно настроить.

Подбирая время задержки и шаг смещения координат, можно добиться плавного эффекта анимации. Ниже представлен небольшой код, в котором вращающийся диск «прогрызает» незатейливый узор (рис. 1.8).

```
# импорт методов из библиотеки turtle
from turtle import *
# функция рисования круга
def ring(x,y,r) :
    pu()
    setx(x)
    sety(y-r)
    pd()
    begin_fill() # начало заливки
```

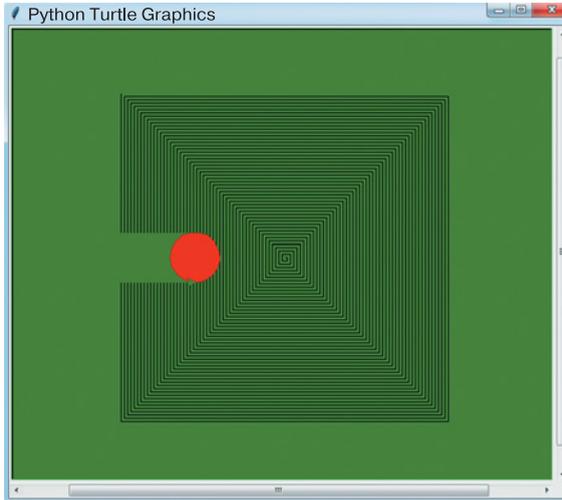


Рис. 1.8. Программирование простейшей анимации

```

tracer(50, 0) # ускоряем анимацию
circle(r)
tracer(1, 0) # замедляем анимацию
end_fill() # конец заливки

# функция рисования узора pattern, придуманная нами
def pattern(s):
# устанавливаем скорость движения курсора
    tracer(2, 5) # эту строчку можно закомментировать
                # и посмотреть изменения работы программы
    dist = s
    for i in range(200):
        fd(dist)
        rt(90)
        dist += s

# функция анимирования
def saw():
    x = 0
    for i in range(200):
# задаем красный цвет
        color(«red»)
# рисуем круг красного цвета
        ring(-200+x, 0, 30)

```

```
# отображаем 10 миллисекунд
    delay(10)
# задаем зеленый цвет
    color(«green»)
# рисуем круг зеленого цвета – цвет фона
    ring(-200+x,0,30)
# отображаем 1 миллисекунду
    delay(1)
# рассчитываем новую координату по X
    x = (i+1)*2

# Основная программа
# задаем размер графического окна 800x600 зеленого
# цвета
screensize(800, 600, «green»)
# рисуем узор с шагом 2

# попробуем незначительно изменить параметр
  в pattern()
pattern(2)
# запускаем анимацию
saw()
```

Экспериментируя с параметрами времени задержки в функциях **delay()**, можно по своему желанию замедлять или ускорять движение красного круга. Подберите такое время, чтобы добиться наиболее плавного движения.

Итоги главы 1

В главе 1 мы познакомились с языком Python и средой программирования IDLE. Оказалось, что синтаксис языка Python прост в понимании и несложен в использовании, однако написание программ на Python требует особой аккуратности и соблюдения отступов строк в коде.

Теперь мы знаем:

1. Адрес официального сайта разработчика языка Python.
2. Порядок установки Python на персональный компьютер.
3. Правила синтаксиса программ на языке Python.
4. Понятие и назначение переменной в Python.

5. Понятие и определение собственной функции в Python.
6. Понятие объектно-ориентированного программирования.
7. Понятие процедурного программирования.
8. Назначение библиотеки и исполнителя **Turtle**.
9. Методы управления пером в **Turtle**.
10. Методы управления окном и анимацией в **Turtle**.

Теперь мы умеем:

1. Скачать и установить Python на компьютер.
2. Установить с помощью пакета **pip** дополнительные библиотеки и пакеты.
3. Создавать и запускать на исполнение программный код Python.
4. С помощью исполнителя **Turtle** писать программы для рисования графических примитивов.
5. Программировать **Turtle**, используя собственные функции.
6. Работать с окном (средой исполнителя) **Turtle**.
7. Программировать простейшее движение графических примитивов с помощью методов управления анимацией пакета **Turtle**.

Глоссарий к главе 1

Здесь собраны основные термины и понятия, а также функции языка Python, встречающиеся в главе 1.

backward(n) | bk(n) | back(n) — функция, перемещает исполнителя **Turtle** назад на n пикселей.

begin_fill() — функция, устанавливает начало заливки в окне **Turtle** указанным цветом.

bgcolor() — функция, задает цвет фона окна **Turtle**, например **bgcolor("orange")**.

bgpic("<file_name.gif>") — функция, загружает фоновое изображение в формате *.gif.

circle(r) — функция, рисует окружность радиусом (r).

def — инструкция, используется для определения функций в Python.

delay(5) — функция, устанавливает или возвращает задержку отрисовки экрана в миллисекундах, приблизительно равную периоду последовательных обновлений холста.

distance(x,y) — функция, возвращает значения расстояния от текущей позиции исполнителя **Turtle** до координаты точки (x, y).

dot(s,c) — функция, рисует точку в текущей позиции, где s — размер знака точки в пикселях, c — название цвета в кавычках, например **"red"**, **"yellow"**, **"dimgray"** и т. д.

- end_fill()** — функция, устанавливает конец заливки указанным цветом.
- fillcolor()** — функция, устанавливает/сообщает цвет заливки.
- forward(n)** | **fd(n)** — функция, перемещает исполнитель вперед на n пикселей.
- goto(x,y)** | **setpos(x,y)** | **setposition(x,y)** — функция, переход в точку с координатами (x, y) .
- home()** — функция, возвращает исполнитель в точку $(0, 0)$.
- IDLE** — встроенная среда разработки Python.
- left(a)** | **lt(a)** — функция, выполняет поворот исполнителя влево на a градусов.
- Numpy** — библиотека, пакет языка Python, предназначенный для работы с большими многомерными массивами и матрицами; содержит в себе большую библиотеку высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.
- pd()** — функция, опустить перо.
- pencolor()** — функция, устанавливает/сообщает цвет пера.
- pensize()** | **width()** — функция, устанавливает толщину пера $(1..10)$.
- pip** — команда, встроенная утилита управления дополнительными пакетами (библиотеками).
- pos()** — функция, возвращает текущую позицию курсора (x, y) .
- pu()** — функция, поднимает перо.
- Python** — высокоуровневый язык программирования общего назначения, разработанный Гвидо ван Россумом в 1991 г. Благодаря своим уникальным особенностям в настоящее время получил широкое распространение среди разработчиков программ.
- python.org** — официальный сайт разработчиков языка Python.
- right(a)** | **rt(a)** — функция, выполняет поворот исполнителя вправо на a градусов.
- screen(1200, 800, "green")** — функция, задает размеры окна и цвет и, если скобки пустые $()$, возвращает размер окна.
- setheading(a)** | **seth(a)** — функция, задает направление на a градусов.
- setx(x)** — функция, смещает в указанную координату x .
- sety(y)** — функция, смещает в указанную координату y .
- speed(s)** — функция, задает скорость рисования s $(0..10)$.
- stamp()** — функция, создает клон «черепашки».
- tracer(n,m)** — функция, включает/выключает анимацию курсора и устанавливает задержку для обновления рисунков; если задано n (может быть использовано для эффекта ускорения/замедления анимации), то выполняется только каждое n -е обновление экрана; если задано m , отличное от 0, устанавливается задержка обновления.
- Turtle** — библиотека, встроенный пакет Python. Представляет собой исполнитель для создания несложных графических рисунков.

undo() — функция, отменяет последнее действие.

xcor() — функция, возвращает текущую позицию курсора по координате x .

ycor() — функция, возвращает текущую позицию курсора по координате y .

Список литературы к главе 1

1. Python Software Foundation // Официальный сайт разработчика Python. URL: <http://python.org>
2. Python. Урок 1. Установка // Devpractice. Разработка программного обеспечения, технологии и наука. URL: <https://devpractice.ru/python-lesson-1-install/>
3. Инструкция по установке и настройке Python 3 // Сообщество Python-программистов. URL: <https://python-scripts.com/install-python>
4. pip — The Python Package Installer. URL: <https://pip.pypa.io/en/stable/>
5. Установка модулей в Python // Сообщество Python-программистов. URL: <https://python-scripts.com/how-to-install-modules-python>
6. Python. Урок 16. Установка пакетов в Python // Devpractice. Разработка программного обеспечения, технологии и наука. URL: <https://devpractice.ru/python-lesson-16-install-packages/>
7. Как устанавливать пакеты в Python с pip и без // Образовательный портал GeekBrains. URL: https://geekbrains.ru/posts/python_packages
8. PEP 8 — Style Guide for Python Code // Python Software Foundation. URL: <https://www.python.org/dev/peps/pep-0008/>
9. PEP 8 — руководство по написанию кода на Python // Python 3 для начинающих. URL: <https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html>
10. PEP 257 — Docstring Conventions // Python Software Foundation. URL: <https://www.python.org/dev/peps/pep-0257/>
11. Документирование кода в Python. PEP 257 // Python 3 для начинающих. URL: <https://pythonworld.ru/osnovy/dokumentirovanie-koda-v-python-pep-257.html>

2

Начало работы с OpenCV-Python

Установка библиотеки OpenCV-Python

OpenCV (Open Source Computer Vision Library)¹ — это библиотека компьютерного зрения с открытым исходным кодом, выпускается под лицензией BSD (Berkeley Software Distribution license) и, следовательно, является бесплатной как для академического, так и для коммерческого использования [1]. OpenCV разработана под языки программирования C++, Python, Java и поддерживает операционные системы Windows, Linux, Mac OS, iOS и Android. Библиотека OpenCV предназначена для создания приложений, выполняющих большие объемы вычислений в реальном времени. Написанная на C/C++ библиотека может использовать аппаратные возможности многоядерной вычислительной платформы. Практическое применение OpenCV — это создание различных систем, использующих в своей работе функции компьютерного зрения. Системы с компьютерным зрением особое значение имеют в робототехнике.

На момент написания данной книги актуальной является OpenCV версии 4.2.0 от 23.12.2019 [2]. В разделе Releases² можно скачать готовый пакет OpenCV под Windows, iOS и Android, а также документацию и архив с исходным кодом. Установка OpenCV для Python имеет свои особенности. В зависимости от того, какая у вас операционная система, процедура установки может несколько различаться. Некоторые способы установки для операционной системы Windows подробно описаны в источниках [3–5].

¹ Open Source Computer Vision Library // Официальный сайт разработчика OpenCV. URL: <https://opencv.org/> (Дата обращения 13.07.2020).

² Releases OpenCV // Официальный сайт разработчика OpenCV. URL: <https://opencv.org/releases.html>

Рассмотрим сначала самый простой способ. После того как вы установили актуальную версию Python, убедитесь, что полный путь к папке с Python не содержит имен папок, написанных на кириллице. В противном случае лучше переустановить Python в другую папку, например «C:\Python\», и затем запустить в командной строке установку OpenCV с помощью утилиты **pip**:

```
pip install opencv-python
```

Если все прошло успешно, проверяем в интерпретаторе Python импорт библиотеки OpenCV и ее версию, по очереди вводя следующие команды:

```
>>> import cv2
>>> print_cv2.version_
```

Однако в ряде случаев (обычно с Windows 7) даже после успешной установки при импорте библиотеки встречается ошибка¹:

```
ImportError: DLL load failed: Не найден указанный
модуль.
```

Рассмотрим порядок установки (примерно как здесь²) OpenCV для Python на компьютер с операционной системой Windows 7, исключая появление данной ошибки (проверялось для 32-битной и 64-битной версий). Для этого вам необходимо следовать приведенным ниже указаниям. Инструкция для Windows 8, 8.1 и 10 выглядит аналогично.

1. Установить бесплатно распространяемый пакет Visual C++ для Visual Studio 2015 [6] в соответствии с разрядностью своей системы. Чтобы это сделать, следует перейти по ссылке³ в Центр загрузки

¹ Форум программистов и системных администраторов Киберфорум. URL: <http://www.cyberforum.ru/python-graphics/thread2337357.html> (Дата обращения 15.12.2018).

² Install OpenCV-Python in Windows // OpenCV-Python Tutorials. URL: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_setup_in_windows/py_setup_in_windows.html#install-opencv-python-in-windows (Дата обращения 20.12.2019).

³ Центр загрузки. Распространяемый пакет Visual C++ для Visual Studio 2015 // Официальный сайт Microsoft. URL: <https://www.microsoft.com/RU-RU/download/details.aspx?id=48145> (Дата обращения 15.12.2018).

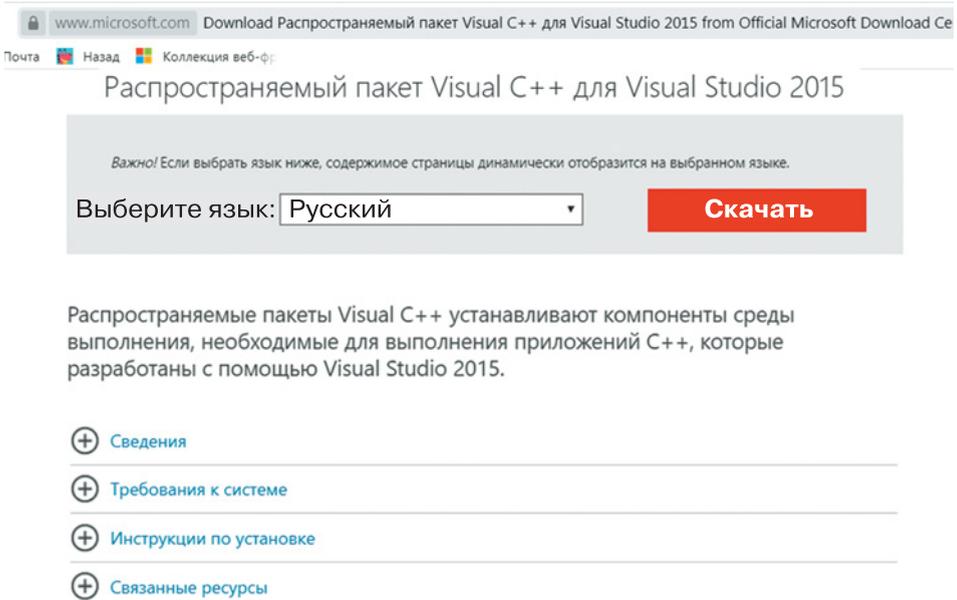


Рис. 2.1. Центр загрузки Microsoft. Распространяемый пакет Visual C++ для Visual Studio 2015

Microsoft (**рис. 2.1**), в списке *Выберите язык* выбрать *Русский* и затем нажать кнопку **Скачать**.

В результате откроется окно загрузки (**рис. 2.2**). Далее выбираем `vc_redist.x64.exe` для 64-битной или `vc_redist.x86.exe` для 32-бит-

Выберите нужную загрузку

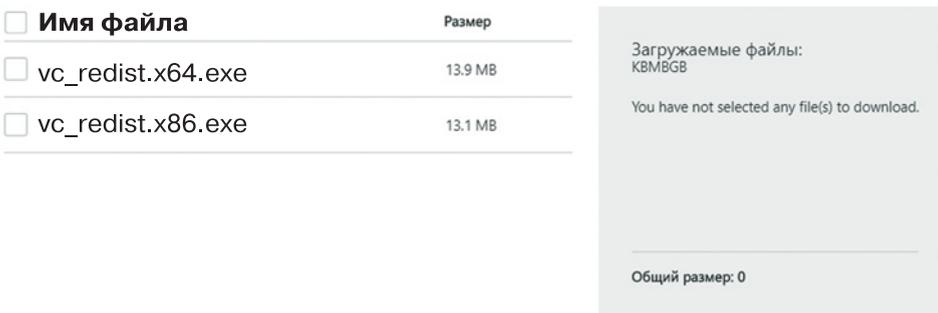


Рис. 2.2. Окно загрузки пакета Visual C++ для Visual Studio 2015

ной Windows 7 и нажимаем кнопку *Next*. Если у вас установлена система Windows другой версии, выберите ее в меню самостоятельно.

По окончании загрузки следует открыть папку (по умолчанию папка *Загрузки*) с загруженным исполняемым файлом пакета Visual C++ для Visual Studio 2015 и запустить его, следуя указаниям мастера установки. Если пакет уже установлен, то мастер выведет окно с сообщением о том, что программа уже установлена.

2. Скачать и установить Python версии 2.7.xx (**Это важно! С версией 3.xx работать не будет!**)¹. Например, можно установить Python 2.7.10 [7]. Для этого по ссылке² скачивается установщик, после чего выполняются все действия, описанные в разделе «Установка Python и дополнительных библиотек» в главе 1.

3. Установить с помощью утилиты `pip` библиотеки Numpy и `Opencv-python`, используя команды:

```
pip install numpy
pip install opencv-python
```

4. Скачать из [8] самораспаковывающийся архив с библиотекой OpenCV для Windows одной из версий 3.1.0–4.1.1. Работоспособность других версий не проверялась. Например, по ссылке³ можно скачать версию OpenCV 3.1.0 и распаковать ее в любую папку. Скачивание начинается автоматически по умолчанию в папку *Загрузки*.

5. Найти в распакованном пакете в папке `\opencv\build\python\cv2\python-2.7\x64` для 64-битной или в папке `\opencv\build\python\cv2\python-2.7\x86` для 32-битной версии установленного Python 2.7.xx файл `cv2.pyd` (**рис. 2.3**).

Найденный файл скопировать поверх в папку, обычно это `C:\Python27\Lib\site-packages\cv2` (**рис. 2.4**).

¹ Хотя в феврале 2020 г. стал доступен релиз Python 3.8.2, многие проекты до сих пор используют старую версию языка. В версиях 2.x и 3.x существуют некоторые отличия в синтаксисе и служебных словах, а также в «скрытой» части — механике работы. Например, деление (a/b) по умолчанию в новой версии дает результат деления без остатка.

² Python Software Foundation // Официальный сайт разработчика Python. URL: <https://www.python.org/ftp/python/2.7.10/python-2.7.10.msi> (Дата обращения 15.12.2018). Здесь 32-битная версия Python, работает в любой версии Windows 7.

³ <http://sourceforge.net/projects/opencvlibrary/files/opencv-win/3.1.0/opencv-3.1.0.exe/download>

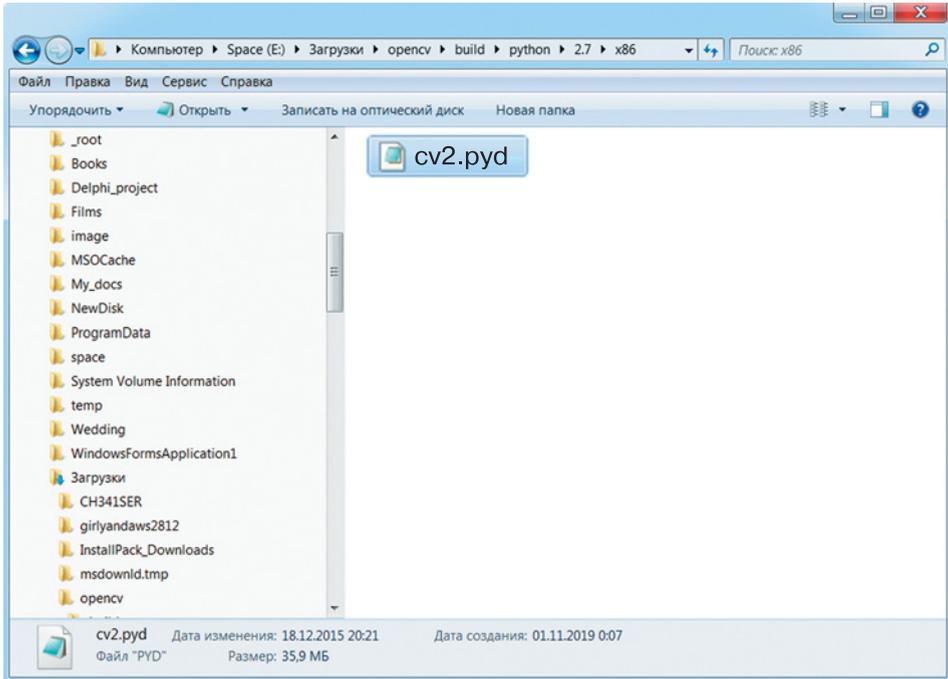


Рис. 2.3. Папка с файлом cv2.pyd для версии Python 32-бит

6. Из меню *Пуск* запустить IDLE (Python GUI) — интерпретатор языка Python 2.7 и проверить отобразившиеся строки:

```
>>> import cv2
>>> print cv2.__version__
3.1.0
>>>
```

На экран будет выведена текущая версия библиотеки OpenCV, в данном случае версия 3.1.0. Это значит, что все отлично и можно приступать к работе!

ПОЯСНЕНИЕ

Официально для OpenCV существуют два типа интерфейсов Python: **cv** и **cv2**. В настоящее время в последних выпусках OpenCV-Python присутствует только cv2. Здесь почти все объекты возвращаются как объекты библиотеки numpy, которая является высокостабильной и быстрой библиотекой обработки массивов данных. Следует

отметить, что применение некоторых функций и их результат будут несколько отличаться в зависимости от используемого интерфейса: `cv` или `cv2`. Это надо учитывать при изучении описания конкретных функций. Иногда в некоторых источниках описание работы функций OpenCV можно найти только применительно к старому интерфейсу `cv`. Здесь мы будем применять *только интерфейс cv2* и, где это будет возможно, указывать на различия в описаниях функций и возвращаемых ими результатов.

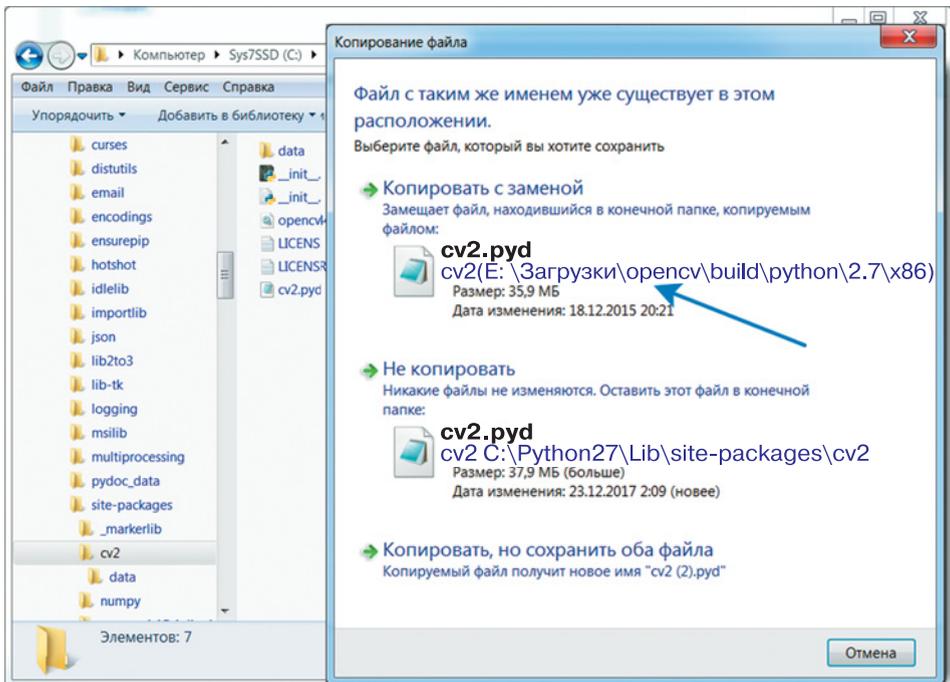


Рис. 2.4. Копирование файла `cv2.pyd` в папку с библиотеками для Python

Основные операции в OpenCV-Python

Основные операции в OpenCV для работы с изображениями связаны с открытием и сохранением файла с изображением, а также с масштабированием, поворотом, кадрированием и отражением по осям. Чтобы научиться выполнять эти операции, предварительно подготовим файл с тестовым изображением. Для этого создадим папку, в которой будем

сохранять все наши скрипты, написанные на Python. Затем скопируем в нее любой графический файл с расширением **.jpg**. Мы взяли фотографию с сотового телефона. Подойдет также фотография с цифровой фотокамеры или web-камеры. Фотографии с сотовых телефонов и цифровых фотокамер имеют большой размер, поэтому их нужно уменьшить, чтобы было удобнее работать с ними. Для этого открываем наш файл в редакторе Paint и сразу видим, что картинка не помещается в окне Paint. Далее нажимаем на вкладке панели инструментов **Главная** кнопку **Изменить размер** (или нажимаем **Ctrl+W**) (**рис. 2.5**). В появившемся окне **Изменение размеров и наклона** выставляем размер **Пиксели** и ставим галочку **Сохранить пропорции**. Затем в поле **По горизонтали** выставляем размер 800 и нажимаем **ОК**.

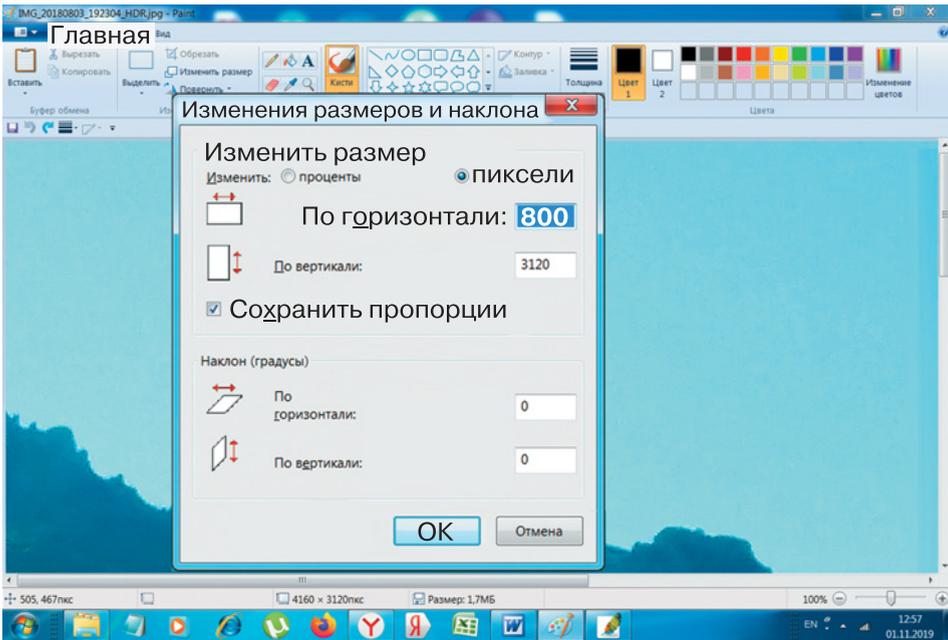


Рис. 2.5. Изменение размеров графического файла с помощью Paint

Теперь остается только сохранить наше изображение под новым именем. Для этого нажимаем клавишу **F12** или выбираем в меню **Сохранить как** → **Изображение в формате JPEG** (**рис. 2.6**). В диалоговом окне **Сохранить как** в поле **Имя файла** вводим с клавиатуры новое имя **testfile**. В поле **Тип файла** выбираем из списка формат **JPEG** и нажимаем кнопку **Сохранить**. Теперь все готово для работы с OpenCV.

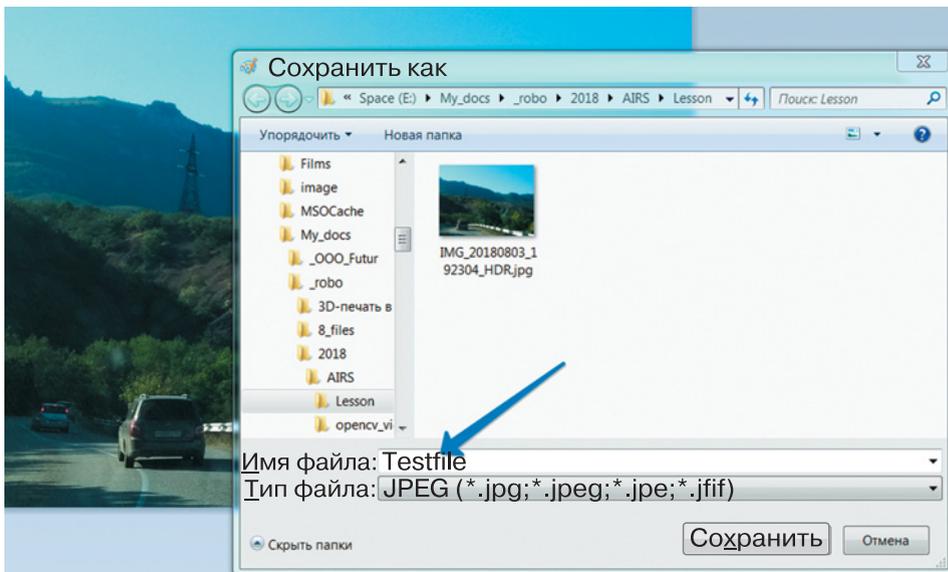


Рис. 2.6. Сохранение измененного графического файла под именем testfile.jpg

Открытие файла с изображением

Для открытия файла с изображением нам необходимо знать две функции:

```
cv2.imread()  
cv2.imshow()
```

Функция **cv2.imread()** считывает в память компьютера массив данных из файла с изображением. В качестве аргумента данной функции необходимо указать имя файла или путь с именем файла. Результат выполнения этой функции присваивается некоторой переменной. Вторая функция **cv2.imshow()** используется для отображения данных графического файла на экране. Эта функция имеет два аргумента: **имя окна**, в котором будет выведено изображение, и **имя переменной**, связанное с отображаемыми графическими данными. В нашем случае это переменная, содержащая результат функции **cv2.imread()**. Ниже представлен программный код для загрузки и отображения графического файла. Не забываем сохранить файл с программой в той же папке, где находится тестовое изображение.

```
# добавим необходимый пакет с opencv  
import cv2  
# присваиваем переменной image изображение из файла  
# testfile.jpg  
image = cv2.imread("testfile.jpg")  
# выводим изображение image на экран в окне Open  
# image  
cv2.imshow("Open image", image)  
# ожидаем нажатия любой клавиши  
cv2.waitKey(0)
```

В строке с командой `cv2.imread("testfile.jpg")` в переменную `image` записываются данные графического файла `testfile.jpg`. А команда `cv2.imshow("Open image", image)` выводит на экран данные из переменной `image` в окне с именем `Open image`. Чтобы после выполнения программы окно с изображением сразу не закрылось, воспользуемся функцией `cv2.waitKey(0)`, которая ожидает нажатия любой клавиши с клавиатуры. После запуска программы на экране должно появиться окно с изображением (рис. 2.7).

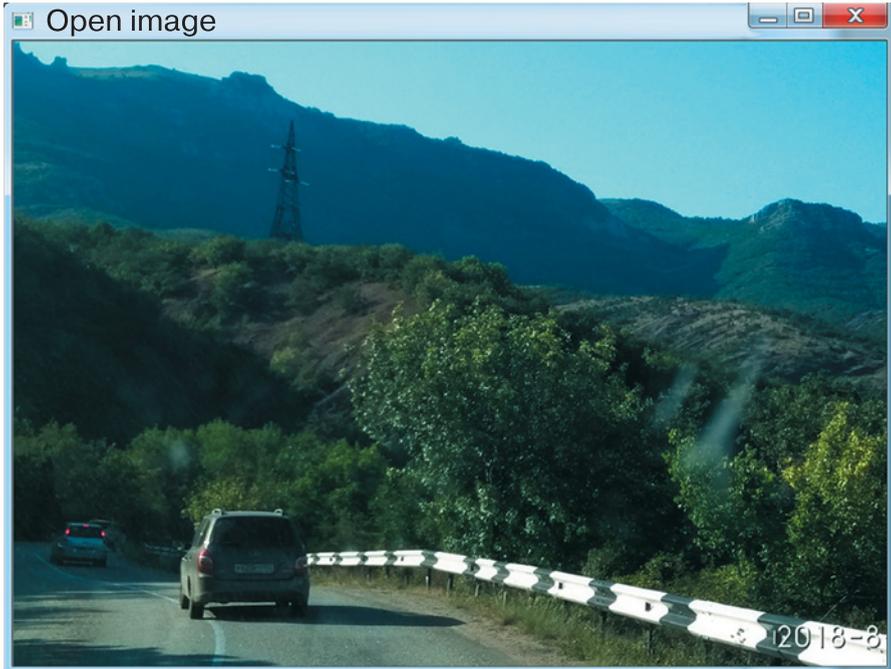


Рис. 2.7. Вывод на экран изображения из файла `testfile.jpg`

Изменение размера изображения

А теперь научимся изменять размер изображения, поскольку бывает, что исходное изображение требуется увеличить или уменьшить. Изменение размера изображения в OpenCV выполняется с помощью функции `cv2.resize()`. Эта функция использует три аргумента и результатом возвращает трансформированное изображение: 1-й аргумент — это имя переменной, связанной с исходным изображением; 2-й аргумент — кортеж из двух элементов (последовательность) с новыми размерами изображения; 3-й аргумент — алгоритм, применяемый для преобразования изображения.

ПОЯСНЕНИЕ

Кортеж — это упорядоченная последовательность из n элементов: (x_1, x_2, \dots, x_n) , где x_i — это **элементы** кортежа. Такой кортеж называют **кортеж длины n** . В информатике кортеж — это упорядоченный набор фиксированной длины. В языке Python есть два похожих типа: **список (list)** и **кортеж (tuple)**.

Список (list)¹ — это структура данных для хранения объектов различных типов. В списке одновременно можно хранить объекты смешанных типов. Размер списка не статичен, его можно изменять. Список по своей природе является изменяемым типом данных.

Кортеж (tuple)² — это неизменяемая структура данных, которая по своему подобию очень похожа на список. Элементы в кортеже изменять нельзя.

Ниже представлен иллюстрирующий код. Впишите нужные комментарии там, где присутствуют знаки «???».

```
# добавляем необходимый пакет с opencv  
import cv2  
# загружаем изображение  
image = cv2.imread("testfile.jpg")  
# задаем ширину изображения в пикселях  
wide = 250  
# вычисляем коэффициент, чтобы сохранить соотношение  
# сторон
```

¹ <https://devpractice.ru/python-lesson-7-work-with-list/>

² <https://devpractice.ru/python-lesson-8-tuple/>

```
f = float(wide) / image.shape[1]
# формируем кортеж (x, y) с размерами изображения
new_size = (wide, int(image.shape[0] * f))
# изменяем изображение и возвращаем его в переменную
# res
res = cv2.resize(image, new_size,
                 interpolation = cv2.INTER_AREA)
# выводим изображение res на экран в окне Resize
# image
cv2.imshow("Resize image", res)
# ???
cv2.imshow("Original", image)
# ???
cv2.waitKey(0)
```

Логика работы программы понятна из комментариев. Изменяя параметр **wide**, можно уменьшать или увеличивать размер конечного изображения. При формировании кортежа с размерами изображения применяется команда **image.shape[0]**. Команда **image.shape[i]** возвращает ширину (при $i = 1$) или высоту (при $i = 0$) исходного изображения **image**.

На **рис. 2.8** представлен результат выполнения программы: исходное изображение в окне **Original** и трансформированное в окне **Resize image**.



Рис. 2.8. Изменение размеров исходного изображения

ЗАДАНИЯ

1. Измените программу, задав фиксированную высоту изображения.
2. Измените программу для уменьшения/увеличения изображения в 2 раза.

Вырезка фрагмента изображения

В системах с компьютерным зрением иногда требуется анализировать не все изображение, а только его часть, где расположены анализируемые объекты, например дорожная разметка, знаки, светофор и т. д. Для этого ненужные части изображения обрезаются. Делается это с помощью срезов.

ПОЯСНЕНИЕ

Под **срезом** понимают некоторый объект, представляющий собой набор индексов (`start`, `stop`, `step`), и метод, предназначенный для получения некоторой части последовательности. Срезы часто используют для представления той или иной последовательности элементов из некоторого списка. Последовательность строится из элементов, начиная с индекса `start` до индекса `stop` (не включая его!) с шагом `step`. По умолчанию значения индексов: `start = 0`, `stop` равен длине исходного списка, `step = 1`. Для создания среза применяют расширенный синтаксис индексирования, используя квадратные скобки «`[]`» и двоеточие «`:`» в качестве разделителя. Например, для одномерного списка `list` [`start: stop: step`].

В нашем случае суть заключается в следующем. Загруженное нами изображение представляет собой сетку (матрицу), состоящую из пикселей, размерностью (**$M \times N$**), где **M** — число строк, которое определяет высоту изображения; **N** — число столбцов, которое определяет ширину изображения. Каждый пиксель имеет свой цвет и координату (**X, Y**). Пиксель в левом верхнем углу изображения имеет координату (0, 0), а в правом нижнем — (**N, M**). Вырезка фрагмента осуществляется путем подбора номеров строк **$Y1, Y2$** (**$Y1 < Y2$**) и номеров

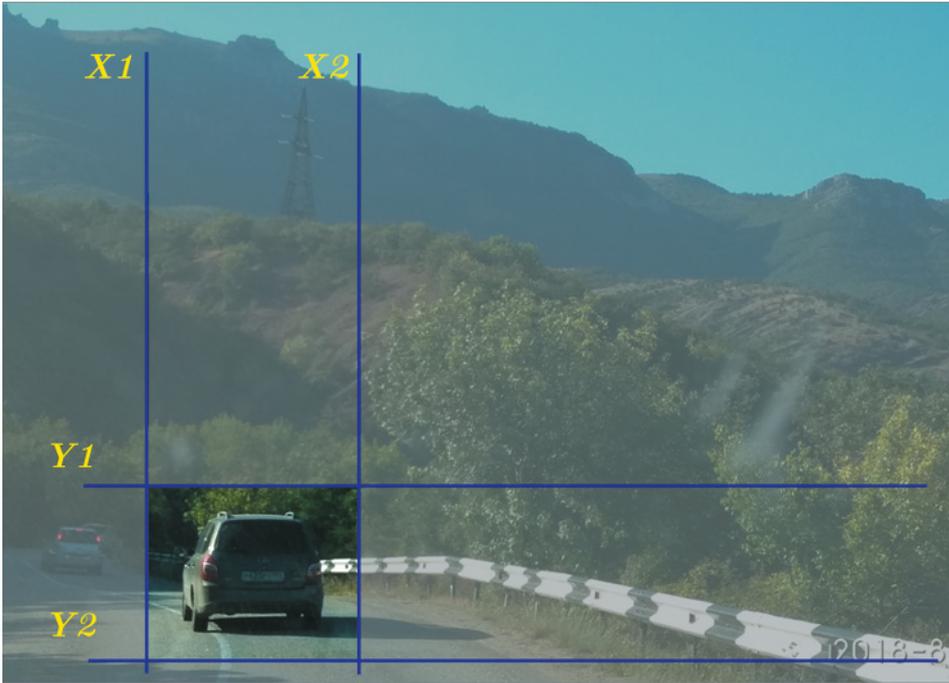


Рис. 2.9. Вырезка фрагмента изображения с помощью срезов

столбцов **X1**, **X2** ($X1 < X2$), между которыми заключен анализируемый объект на изображении (**рис. 2.9**).

Рассмотрим программную реализацию вырезки фрагмента изображения. Для этого воспользуемся нашим файлом с тестовым изображением (`testfile.jpg`), на котором запечатлен объект — автомобиль, движущийся по трассе. Далее следует простой алгоритм. Уже известным нам способом считываем данные исходного изображения в некоторую переменную. Затем, используя срезы, полученный фрагмент исходного изображения присваиваем новой переменной. После чего с помощью функции `cv2.imshow()` отображаем данные обеих переменных. Ниже приведен программный код для выполнения этой операции.

Вместо знаков «???» впишите правильные комментарии.

```
import cv2
# ???
image = cv2.imread("testfile.jpg")
# вырежем фрагмент изображения, используя срезы
# y1:y2, x1:x2
```

```
crop = image[420:550, 150:280]
# ???
cv2.imshow("Original", image)
# выводим фрагмент изображения crop в окне CropImage
cv2.imshow("CropImage", crop)
# ???
cv2.waitKey(0)
```

В результате на экране будут отображены два окна: исходное изображение и фрагмент исходного изображения (рис. 2.10).

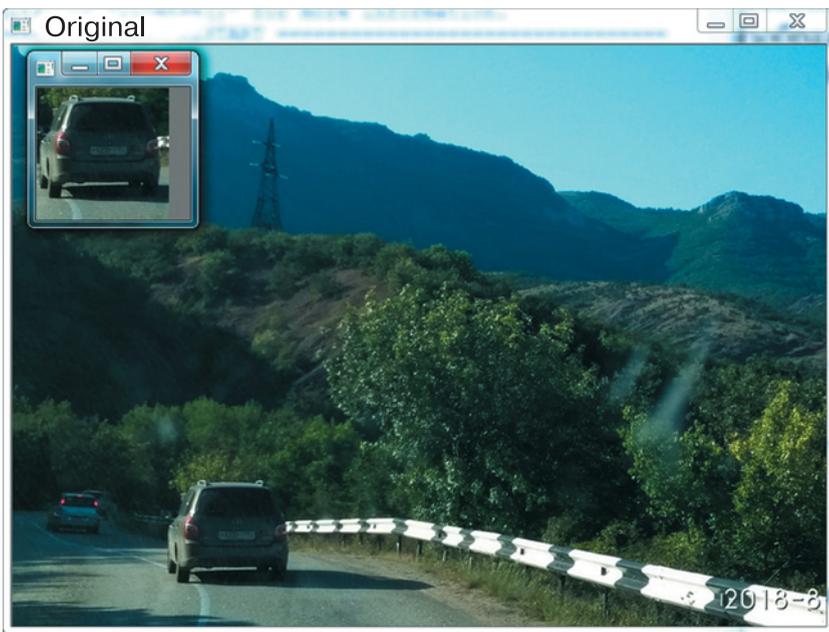


Рис. 2.10. Вырезка фрагмента изображения

ЗАДАНИЯ

1. Подберите срезы так, чтобы в вырезанный фрагмент попали автомобили или другие объекты на заднем плане вашего рисунка.
2. Используя программу из прошлого урока, выведите на экран изображение вырезанного фрагмента, увеличенное в 3 раза.

Поворот изображения

Для поворота изображения необходимо применить две функции:

```
cv2.getRotationMatrix2D()  
cv2.warpAffine()
```

Функция **cv2.getRotationMatrix2D()** возвращает двумерный объект, в котором будет размещено преобразованное изображение (в случае трансформации и поворота). В этой функции используется три аргумента:

- первый аргумент — координаты точки, относительно которой будет осуществлен поворот и изменение размера изображения;
- второй аргумент — угол поворота в градусах;
- третий аргумент — коэффициент трансформации, который может быть и меньше 1.

Вторая функция **cv2.warpAffine()** выполняет поворот изображения и возвращает новое изображение. Эта функция также использует три аргумента:

- первый аргумент — имя переменной исходного изображения;
- второй аргумент — имя переменной двухмерного объекта для преобразованного изображения (см. выше);
- третий аргумент — список с размерами выходного изображения.

Логика программы такова. После считывания исходного изображения получаем его размер: высоту и ширину в виде кортежа. Затем определяем координаты точки, относительно которой будет выполняться поворот изображения. Далее с помощью функции **cv2.getRotationMatrix2D()** подготавливается объект с параметрами преобразования исходного изображения. Результат выполнения функции записывается в некоторую переменную. Следующий шаг — с помощью функции **cv2.warpAffine()** выполняется преобразование исходного изображения, а результат присваивается соответствующей переменной, данные из которой отображаем с помощью функции **cv2.imshow()**. Ниже представлен пример программного кода для иллюстрации операции поворота изображения на 180° .

```
import cv2  
# загружаем изображение  
image = cv2.imread("testfile.jpg")
```

```
# получаем размеры исходного изображения для поворота

```

В примере использована функция `image.shape[:2]`, которую мы уже применяли. Здесь эта функция возвращает размеры изображения (высота × ширина) в виде кортежа `(h, w)`. После выполнения программного кода на экран будет выведено окно с повернутым изображением (рис. 2.11).

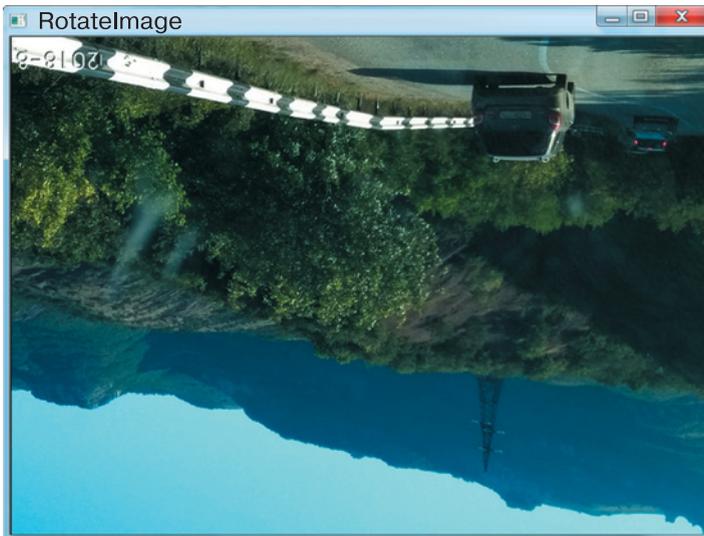


Рис. 2.11. Поворот изображения на 180°

ЗАДАНИЯ

1. Поверните исходное изображение на 45° , 90° , 120° .
2. Выведите на экран уменьшенное перевернутое изображение.
3. Вырежьте произвольный фрагмент изображения и выполните его поворот.

Зеркальное отражение изображения по осям

В OpenCV для любого изображения можно сделать зеркальную копию. Создание зеркальных копий изображений выполняется с помощью функции `cv2.flip()`. У этой функции имеются два аргумента:

- первый аргумент — имя переменной с исходным изображением;
- второй аргумент определяет ось, относительно которой производится отражение.

Для второго аргумента могут быть использованы следующие значения: 0 — отражение по горизонтали (вокруг оси x), 1 — отражение по вертикали (вокруг оси y), (-1) — по вертикали и по горизонтали.

Результат выполнения функции — отраженное изображение, которое можно сохранить в новой переменной и вывести на экран с помощью функции `cv2.imshow()`. Логика работы программы здесь очень простая, поэтому сразу перейдем к изучению программного кода. Ниже представлен пример программного кода для иллюстрации операции зеркального отражения изображения по вертикальной оси.

Там, где стоят знаки «??», впишите правильный комментарий.

```
import cv2
# загружаем изображение и отображаем его
image = cv2.imread("testfile.jpg")
# отражаем изображение по вертикальной оси
flipped = cv2.flip(image,1)
# ???
cv2.imshow("Original", image)
# ???
cv2.imshow("Flip image", flipped)
# ???
cv2.waitKey(0)
```

Как видно из первых комментариев, сначала выполняется считывание исходного изображения, а после этого с помощью функции `cv2.flip()` выполняется его преобразование, результат которого записывается в переменную `flipped`. На **рис. 2.12.** представлен результат выполнения программой зеркального отражения.

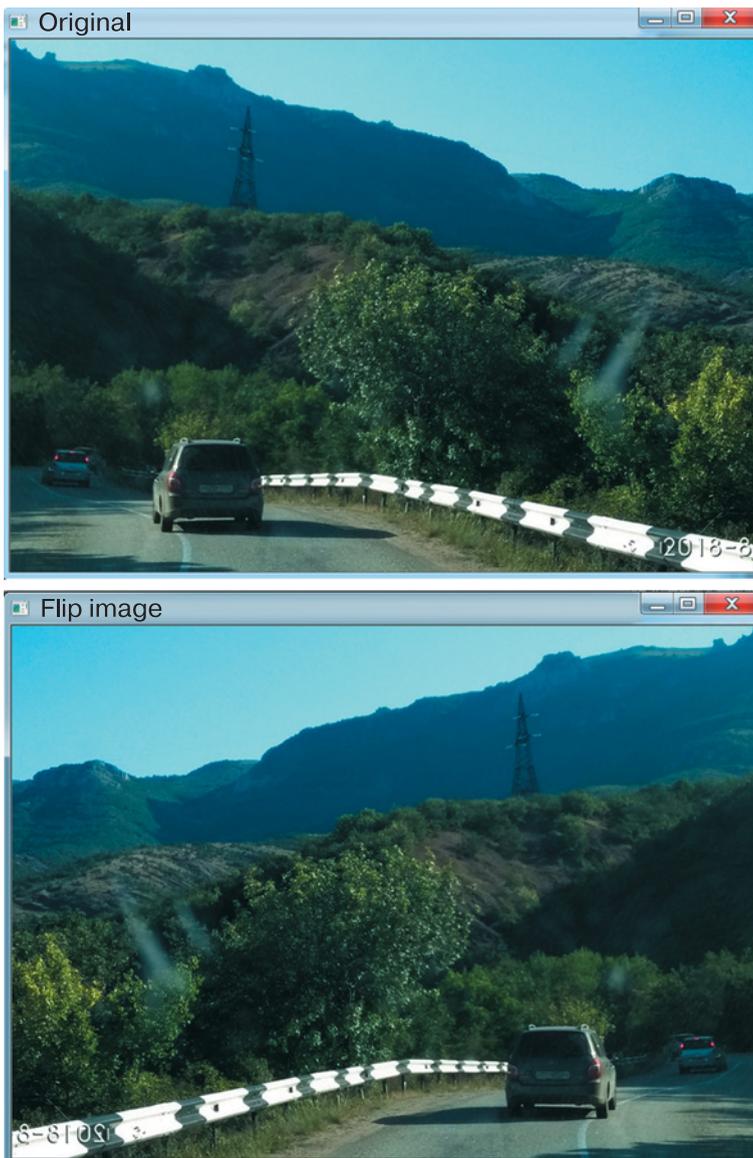


Рис. 2.12. Зеркальное отражение изображения

ЗАДАНИЯ

1. Выполните оставшиеся способы зеркального отражения.
2. Вырежьте произвольный фрагмент изображения и выполните его зеркальное отражение одним из способов.

Сохранение изображения в файл на локальный диск

Результаты операций с изображениями, которые мы рассмотрели выше, выводились в отдельные окна на экран. Иногда эти результаты необходимо сохранить на жесткий диск в каком-нибудь графическом формате.

Чтобы сохранить преобразованное изображение в графический файл, применяют функцию `cv2.imwrite()`. Для этой функции надо указать два аргумента:

- первый аргумент — это путь с именем файла или имя файла (например, `<имя файла>.jpg`, `< имя файла >.png` или `<путь>\<имя файла>.jpg` и т. д.), в котором будет записано преобразованное изображение;
- второй аргумент — это имя переменной того изображения, которое требуется сохранить.

В зависимости от указанного расширения в имени файла, OpenCV сама определяет тип графического изображения и выполняет все операции по преобразованию формата. Ниже приводится пример программного кода, сохраняющего зеркальную копию исходного изображения в формате `.png`.

Внимательно изучите программу и впишите вместо знаков «???» правильные комментарии.

```
import cv2
# ???
image = cv2.imread("testfile.jpg")
# ???
flip_image = cv2.flip(image,1)
# ???
cv2.imshow("Original", image)
# ???
```

```
cv2.imshow("Flip image", flip_image)
# запишем изображение на диск в формате .png
cv2.imwrite("flip.png", flip_image)
cv2.waitKey(0)
```

В результате выполнения кода в папке с файлами программы и исходного изображения появится новый файл `flip.png` (рис. 2.13).

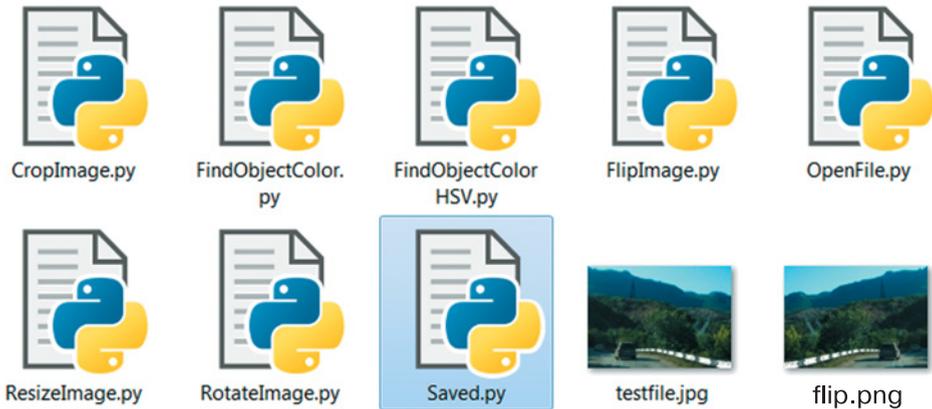


Рис. 2.13. Результат сохранения файла

ЗАДАНИЯ

1. Сохраните фрагмент изображения (см. раздел «Вырезка фрагмента изображения») в файл формата **.jpg**.
2. Сохраните одно из преобразованных изображений (см. раздел «Поворот изображения») в файл формата **.png**.

Поиск объекта по цвету

В системах с компьютерным зрением одной из ключевых задач является идентификация объектов, попавших в поле зрения фото- или видекамеры. Наиболее эффективным методом поиска объекта считается определение объекта по его цвету. Это делается с помощью цветовой маски. Цветовая маска при наложении на исходное изобра-

жение позволяет отфильтровать цвета согласно выбранному цветовому диапазону. Области изображения, которые не соответствуют указанному диапазону цветов, отображаться не будут. После применения цветовой маски эти области будут окрашены в черный цвет, а все, что соответствует цветовому диапазону, — в белый цвет. Это сильно упрощает задачу поиска объектов. Таким образом, подбирая нужный диапазон цветов, можно вывести на экран только искомый объект или группу объектов.

Рассмотрим задачу поиска объекта по цвету на графическом изображении. Для этой задачи мы взяли изображение из Интернета¹ — ярко-желтый теннисный мяч на траве. Давайте попробуем разобрать решение задачи на приведенном ниже программном коде.

```
import cv2
# загрузка изображения
image = cv2.imread('YellowBall.jpg')
# ???
cv2.imshow("Original", image)
# задаем границы диапазона:
# нижнюю
low_color = (0,0,150)
# и верхнюю
high_color = (255,255,255)
# наложение цветовой маски на исходное изображение,
# результат присваиваем переменной only_object
only_object = cv2.inRange(image, low_color,
                          high_color)
# вывод отфильтрованного изображения на экран
cv2.imshow('only object', only_object)
cv2.waitKey(0)
```

После загрузки изображения (функция `cv2.imread()`) требуется подобрать нижнюю и верхнюю границы цветового диапазона для искомого объекта. Поскольку цвет пикселей изображения задается смещением компонент красного, зеленого и синего цветов (Red-Green-Blue — цветовое RGB-пространство), границы диапазона будут определяться совокупностью трех значений каждой компоненты по шкале от 0 до 255.

¹ https://cdn.pixabay.com/photo/2016/06/23/10/41/tennis-ball-1475072_960_720.jpg

Наш объект на изображении представляет собой группу пикселей различных желтых оттенков, поэтому в диапазоне между нижней и верхней границей будут преобладать компоненты зеленого и синего цветов. Однако красная компонента тоже будет присутствовать, так как мячик все-таки не идеально желтого цвета. В программе границы диапазона задаются в переменных **low_color**, **high_color**. В данном случае используется цветовое пространство **BGR**: (Синий (Blue), Зеленый (Green), Красный (Red)) по шкале от 0 до 255.

Затем с помощью функции **cv2.inRange()** на исходное изображение (переменная **image**) накладывается цветовая BGR-маска и отфильтровываются все цвета вне диапазона **low_color**, **high_color**. Функция **cv2.inRange()** принимает на входе цветное изображение и возвращает на выходе черно-белое изображение, где белыми пикселями отрисованы области, цвета которых попадали в заданный диапазон.

На **рис. 2.14** показан результат работы программы. Как видно из рисунка, кроме теннисного мячика, присутствует большое количество белых вкраплений. Они тоже попали в наш цветовой диапазон.

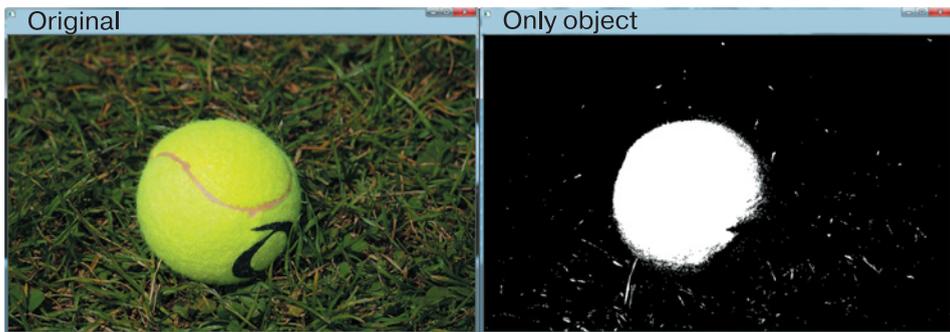


Рис. 2.14. Результат поиска с использованием цветовой BGR-маски

ЗАДАНИЯ

1. Загрузите произвольное изображение с дорожными знаками и подберите параметры **low_color**, **high_color** для выделения одного из знаков или каких-либо ярких объектов.
2. Подберите и запишите параметры **low_color**, **high_color** для различных дорожных знаков, ярких объектов из различных изображений. Выполните вывод на экран.

Цветовые пространства

Эксперименты с поиском объектов по цвету показали, что далеко не всегда удастся выделить только нужный объект. Часто вместе с объектом на изображении выделяется множество лишних точек — так называемый шум. Появление шумов на изображении затрудняет выполнение дальнейшего анализа картинку, поэтому необходимо максимально избавиться от шумов, определив причины их возникновения. Для этого рассмотрим более подробно понятие **цветовое пространство**.

В предыдущем разделе мы выполняли поиск объекта в цифровом RGB (BGR)-пространстве. Обратимся к **рис. 2.15**:

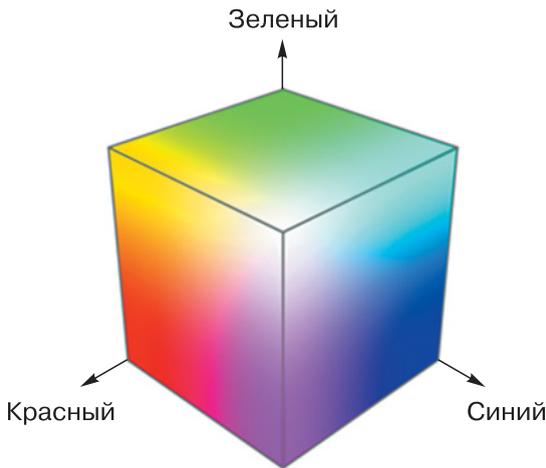


Рис. 2.15. Цветовое RGB (BGR)-пространство (Источник: <https://arboook.com/kompyuternoe-zrenie/nahodim-tsvetnoj-predmet-v-kadre-s-pomoshhyu-opencv-3-python/>)

Как видно из рисунка, RGB-пространство представляет собой куб с длиной ребра 256, в котором любой цвет задается координатами соответствующей точки на этом кубе. Широкое распространение такой модели формирования цвета объясняется тем, что на экране телефона или монитора компьютера изображение формируется за счет точек трех цветов — синего, зеленого и красного.

Функция `cv2.inRange()` в качестве аргументов использует наборы из двух цветов — точек на данном кубе. В таком случае в выделенный диапазон, который удовлетворяет нашим критериям, попадет цветовая область, схематично представляющая собой **прямоугольный RGB-параллелепипед** (рис. 2.16).

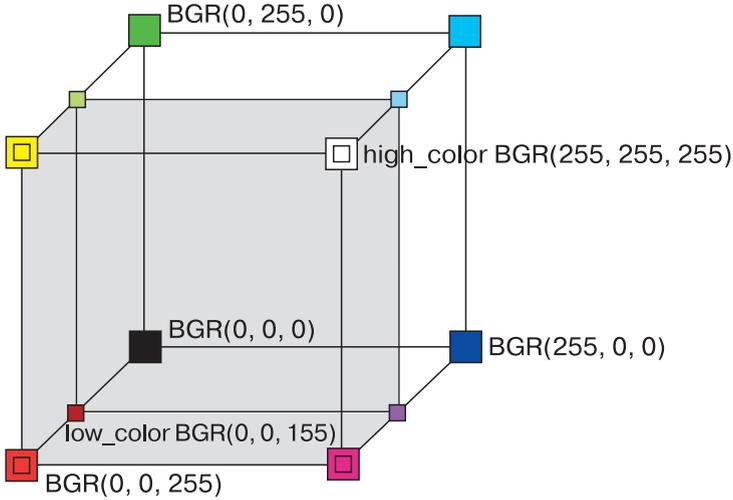


Рис. 2.16. Диапазон `low_color`, `high_color` в пространстве цветов BGR

Таким образом, наша функция находит вложенный цветовой параллелепипед в полном кубе RGB. И если требуется найти объект с оттенками желтого цвета, то размещаем вложенный цветовой параллелепипед как можно ближе к желтой области. К сожалению, при таком размещении в указанный диапазон попадают и другие оттенки цветов, из-за того что грани параллелепипеда параллельны граням основного куба RGB. В результате на изображении, кроме пикселей искомого объекта, появляются пиксели других цветов, не относящиеся к объекту.

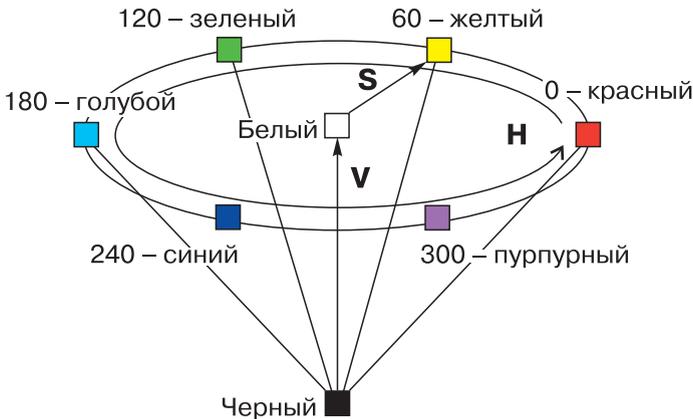


Рис. 2.17. Цветовое пространство HSV (HSB)

Одно из существующих решений данной проблемы — это переход в другое цветовое пространство. Наиболее эффективным здесь является **HSV (HSB)-пространство**. В данной модели основными координатами являются цветовой тон (Hue), насыщенность (Saturation) и значение или яркость (Value или Brightness). Под *цветовым тоном* понимается именно цвет, измеряется от 0 до 360. *Насыщенность* характеризует близость цвета к белому (розовый ближе к белому, чем красный), измеряется в процентах. А параметр *значение* представляет как общую яркость точки — чем меньше, тем темнее, также измеряется в процентах. Схематично эту модель можно изобразить в виде перевернутого конуса (**рис. 2.17**).

Поиск объекта в цветовом пространстве HSV

Возвращаемся к нашей задаче поиска объекта по цвету. Диапазон выбранных цветов будет представлять область внутри HSV-конуса, ограниченную нижними и верхними значениями параметров H, S и V (**рис. 2.18**).

Из рисунка видно, что в заданных пределах границы главных атрибутов цветового фильтра в модели HSV — тон (H), насыщенность (S) и значение (V)¹ — определяются просто и вполне по-

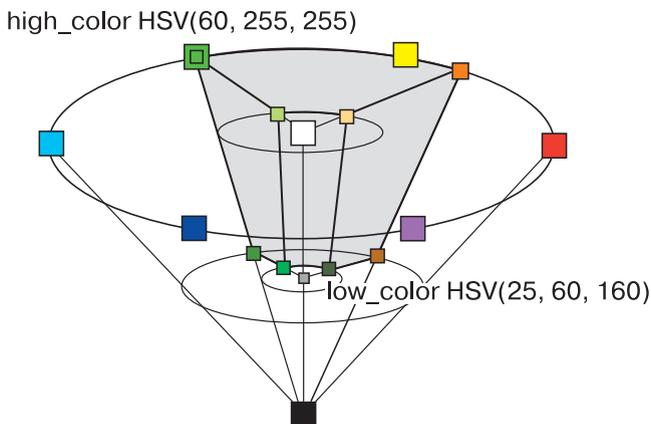


Рис. 2.18. Цветовой диапазон `low_color`, `high_color` в пространстве цветов HSV. Значения компонентов H, S, V указаны в соответствии со шкалой OpenCV3 (см. ниже)

¹ В OpenCV3 данное цветовое пространство имеет обозначение HSV.

нятно. Границы H определяют основной диапазон цветов пикселей искомого объекта, S — насколько блеклые и насыщенные цвета пикселей объекта, а V — принадлежность к области темных и ярких оттенков.

Чтобы найти наш теннисный мячик, переведем изображение с ним в цветовое пространство HSV и попытаемся найти мячик на картинке. Ниже представлен пример программного кода:

```
import cv2
# загрузка изображения
image = cv2.imread('YellowBall.jpg')
# ???
cv2.imshow("Original", image)
# конвертируем исходное изображение в HSV,
# результат присваиваем переменной hsv_img
hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
# нижняя граница — это темный ненасыщенный цвет
color_low = (25,100,175)
# верхняя граница — это яркий насыщенный цвет
color_high = (35,255,255)
# наложение цветовой маски на HSV-изображение,
# результат присваиваем переменной only_object
only_object = cv2.inRange(hsv_img, color_low,
                           color_high)
# вывод отфильтрованного изображения на экран
cv2.imshow('color_hsv', only_object)
cv2.waitKey(0)
```

Здесь мы применили новую функцию `cv2.cvtColor()`. Она конвертирует исходное изображение из одного цветового пространства в другое и возвращает сконвертированное изображение. В качестве аргументов используются два параметра: имя переменной исходного изображения и значение направления конвертации, которое задается специальным кодом, например: `cv2.COLOR_BGR2RGB`, `cv2.COLOR_BGR2HSV` и т. д.

Для поиска цветного объекта необходимо подобрать диапазон `color_low = (H1, S1, V1)`, `color_high = (H2, S2, V2)`. С помощью значений $H1$, $H2$ ($H1 < H2$) определяются цветовые оттенки, в которые окрашен искомый объект (в OpenCV3 шкала цветового

тона (H) лежит в значениях $0...180^1$, шкалы (S) и (V) — в значениях $0...255$). Для нашего мячика желтого цвета значение $H = 30$ (половина от значения на рис. 2.17). Учитывая наличие оттенков, расширим диапазон от 25 до 35 — это будут значения $H1$ и $H2$. Поскольку мяч на картинке яркого и насыщенного цвета, то значения для $S2$ и $V2$ примем максимальные — 255. Значения для $S1$ и $V1$ примем равными 0. Далее запускаем программу и визуально оцениваем ее работу. Сужая диапазоны за счет увеличения значений $S1$ и $V1$ и уменьшения значений $S2$ и $V2$, а также расширяя или сужая интервал между $H1$ и $H2$, добиваемся удаления шума в изображении. Результат работы программы представлен на **рис. 2.19**.

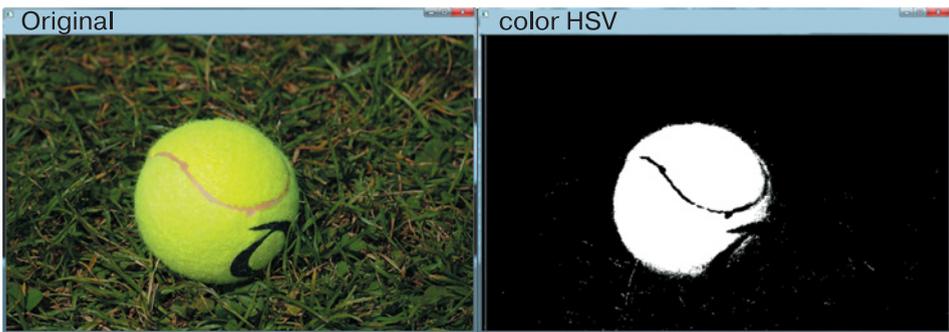


Рис. 2.19. Результат поиска с использованием цветовой HSV-маски

Из рисунка видно, что, по сравнению с предыдущими результатами (см. рис. 2.14.), здесь зашумленность гораздо ниже. Первый шаг в задаче идентификации объектов, попавших в поле зрения фотокамеры, мы сделали!

ЗАДАНИЕ

1. Загрузите изображения с дорожными знаками из предыдущего задания и подберите параметры **color_low**, **color_high** для выделения одного из знаков или каких-либо ярких объектов.

¹ Для кодирования значений — H, S, V, точно так же, как и для RGB (BGR) цветового пространства, используется по одному байту на каждую компоненту, что ограничивает пределы от 0 до 255. Поскольку в диаграмме число значений для компоненты H больше 255, для удобства пересчета пределы H ограничены от 0 до 180 и ее также можно кодировать одним байтом.

Определение координат найденного объекта

Итак, мы научились находить объекты на картинке с фотокамеры с помощью цветowych фильтров. Они получаются в виде черно-белых изображений, на которых искомый объект окрашен в белый цвет, а окружающий его фон — в черный.

Следующий шаг — определение положения объекта на картинке. Это можно сделать, вычислив координаты искомого объекта. Поначалу эта задача может поставить в тупик, потому что объект, который мы хотим найти, — не точка на картинке, а целое пятно белого цвета и неправильной формы (см. рис. 2.14 и 2.19), состоящее из множества точек. Каждая точка имеет свою координату. Оперировать непосредственно таким множеством координат не очень удобно. Однако не стоит расстраиваться, у разработчиков OpenCV есть много хороших идей! Для определения координат нашего «пятна», то есть объекта, в библиотеке OpenCV применяется функция вычисления моментов `cv2.moments()`.

ПОЯСНЕНИЕ

Момент изображения — это суммарная характеристика пятна, представляющая собой сумму всех точек этого пятна. Существует множество подвидов моментов, характеризующих разные свойства изображения. Например, момент нулевого порядка `m00` — это количество всех точек, составляющих пятно. Момент первого порядка `m10` представляет собой сумму *x*-координат точек, а `m01` — сумму их *y*-координат. Имеются также моменты `m11`, `m20`, `m02`, `m22` и т. д.

Для работы функции `cv2.moments()` необходимо указать два аргумента:

- первый аргумент — имя переменной, с которой связано анализируемое изображение;
- второй аргумент — двоичное число, по значению которого определяется, как алгоритм должен вычислять вес каждой точки.

Мы в нашей задаче будем применять функцию `cv2.moments()` к изображению, которое нам возвращает функция `cv2.inRange()`. А это черно-белое изображение, точки которого, кроме черного и белого цвета, могут иметь градации серого цвета. Так вот, если значение второго аргумента для функции `cv2.moments()` равно 1, то вес

всех точек с цветом, отличным от нуля, будет равен 1, вес черной точки будет равен 0, а белой точки — 255.

Для решения нашей задачи достаточно будет воспользоваться значениями моментов нулевого и первого порядков. Затем, поделив поочередно сумму x -координат ($m10$) и y -координат ($m01$) всех точек пятна на число его точек ($m00$), получим некоторые усредненные координаты нашего пятна по осям x и y . Поскольку пятно является непосредственным отображением нашего объекта на отфильтрованном изображении, мы тем самым получаем координаты объекта на рисунке. Полученные таким образом координаты можно использовать, чтобы подписать найденный объект. В этом нам поможет еще одна функция из библиотеки OpenCV: `cv2.putText()`. А вот как это будет выглядеть в программном коде:

```
import cv2
# загрузка изображения
image = cv2.imread('YellowBall.jpg')
# конвертируем исходное изображение в HSV,
# результат присваиваем переменной hsv_img
hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
# нижняя граница - это темный ненасыщенный цвет
color_low = (25,60,160)
# верхняя граница - это яркий насыщенный цвет
color_high = (60,255,255)
# наложение цветовой маски на HSV-изображение,
# результат присваиваем переменной only_object
only_object = cv2.inRange(hsv_img, color_low,
                           color_high)
# вычисляем моменты отфильтрованного HSV-изображения
moments = cv2.moments(only_object, 1)
# вычисляем сумму  $x$ -координат всех точек пятна
x_moment = moments['m01']
# вычисляем сумму  $y$ -координат всех точек пятна
y_moment = moments['m10']
# вычисляем общее число всех точек пятна
area = moments['m00']
# вычисляем среднее значение координаты  $x$  объекта
x = int(x_moment / area)
# вычисляем среднее значение координаты  $y$  объекта
y = int(y_moment / area)
# выводим надпись "Yellow ball" на изображение
```

```
cv2.putText(image, "Yellow ball", (x,y),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2)
# Выводим координаты объекта
cv2.putText(image, "%d, %d" % (x,y), (x,y+30),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2)
# Выводим картинку в окне found
cv2.imshow('found', image)
cv2.waitKey(0)
```

В результате выполнения программы на экране получим картинку, показанную на **рис. 2.20**.

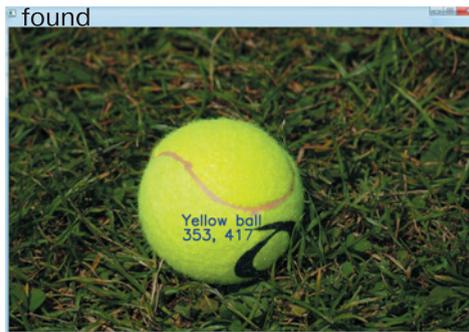


Рис. 2.20. Обозначение найденных объектов

Значения цветового фильтра те же, что и в предыдущей программе, поскольку картинку и искомый объект мы не меняли. Следует обратить внимание на большое количество аргументов у функции **cv2.putText()**. Сделаем небольшое пояснение, чтобы было понятно, как она работает. Первый вызов этой функции в программе выводит на изображении **image** (исходное изображение) надпись **Yellow ball** (текстовые значения помещаем в кавычки) по рассчитанным нами ранее координатам **(x,y)**. Для надписи используется шрифт **cv2.FONT_HERSHEY_SIMPLEX**, размер — **1**, цвет — синий в комбинации трех чисел **(255,0,0)**, толщина линий букв — **2**. Вторым вызовом в следующей строке выводятся значения координат объекта в виде текста с помощью инструкции **"%d, %d" % (x,y)**, где **(x,y)** — численные значения координат. При этом сама надпись, чтобы не было наложения с предыдущей, выводится по координатам **(x,y+30)**.

Отображение видео в OpenCV-Python

В предыдущих разделах мы познакомились с основами обработки данных графических файлов. Теперь перейдем к следующему этапу — обработке данных из оцифрованного видео. На практике в системах с компьютерным зрением графическая информация поступает видеопотоком — в виде сплошной последовательности изображений (кадров). Источником видеоданных может быть видеокамера, которая осуществляет съемку в реальном времени, или видеофайл, в который эти данные были предварительно записаны. (В наших задачах мы будем пользоваться видеоданными с обычной web-камеры или из видеофайла формата ***.avi**).

ПОЯСНЕНИЕ

Формат AVI (Audio Video Interleave) — это медиаконтейнер, использованный впервые разработчиками Microsoft в пакете Video for Windows. Файл AVI может содержать видео- и аудиоданные, сжатые с помощью разных комбинаций видео- и аудиокодеков, что позволяет синхронно воспроизводить видео со звуком. Под **кодеком** понимается алгоритм записи потоков видео или аудио. С полной документацией по формату AVI можно ознакомиться [здесь](#)¹.

Отображение видеоданных с web-камеры

Рассмотрим программу отображения видео в режиме захвата видеопотока с web-камеры. Ниже приведен пример программного кода отображения видео с web-камеры.

```
import cv2
# связываем видеопоток камеры с переменной capImg
capImg = cv2.VideoCapture(0)
# запускаем бесконечный цикл, чтобы следить
# в реальном времени
```

¹ AVI RIFF File Reference (Windows CE 5.0)/Microsoft Docs. URL: [https://docs.microsoft.com/en-us/previous-versions/windows/embedded/aa451196\(v=msdn.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/windows/embedded/aa451196(v=msdn.10)?redirectedfrom=MSDN)

```
while (True) :  
# получаем кадр из видеопотока,  
# кадры по очереди считываются в переменную frame  
    ret, frame = capImg.read()  
# показываем кадр в окне 'Video'  
    cv2.imshow('Video', frame)  
# организуем выход из цикла по нажатию клавиши,  
# ждем 30 миллисекунд нажатие, записываем код  
# нажатой клавиши  
    key_press = cv2.waitKey(30)  
# если код нажатой клавиши совпадает с кодом  
# «q»(quit - выход),  
    if key_press == ord('q'):  
#         то прервать цикл while  
        break  
# освобождаем память от переменной capImg  
capImg.release()  
# закрываем все окна opencv  
cv2.destroyAllWindows()
```

Как видно из комментариев к строкам, алгоритм работы программы несложен. Вначале, как обычно, импортируется модуль OpenCV, затем с помощью функции **cv2.VideoCapture(0)** видеоданные связываются с переменной **capImg**. Значение «0» в аргументе функции обозначает номер подключенной камеры. Нумерация начинается с нуля. Далее запускается бесконечный цикл, где полученные из видеопотока кадры (**frame**) отображаются в окне «**Video**» с помощью функции **cv2.imshow()**. Завершение работы программы осуществляется по нажатию определенной клавиши с помощью уже известной нам функции **cv2.waitKey()**. Наверное, вы уже заметили, что здесь значение аргумента этой функции другое. Если это значение больше нуля — оно определяет время ожидания нажатия клавиши в миллисекундах. При значении, равном нулю, — нажатие клавиши ожидается бесконечно. Также отметим, что функция **cv2.waitKey()** не только ожидает нажатия клавиши, но и возвращает код нажатой клавиши. В нашем случае это будет код ASCII.

ПОЯСНЕНИЕ

ASCII (American Standard Code for Information Interchange) — название кодировочной таблицы, в которой распространенным пе-

чатным и непечатным символам сопоставлены числовые коды. Таблица была разработана и стандартизована в США в 1963 г. Таблица ASCII определяет коды для символов: десятичных цифр, латинского алфавита, национального алфавита, знаков препинания, управляющих символов. Таблицу ASCII кодов можно посмотреть здесь¹.

Код нажатой клавиши мы запишем в переменную `key_press` и сравним его с кодом клавиши «q», воспользовавшись функцией `ord('q')`, встроенной в Python. Функция `ord()` возвращает код указанного символа. Если это условие сравнения верно, программа завершит свою работу, прервав цикл инструкцией `break`. Оператор `break` позволяет прервать выполнение любого цикла. На **рис. 2.21** показан результат выполнения представленного выше кода.

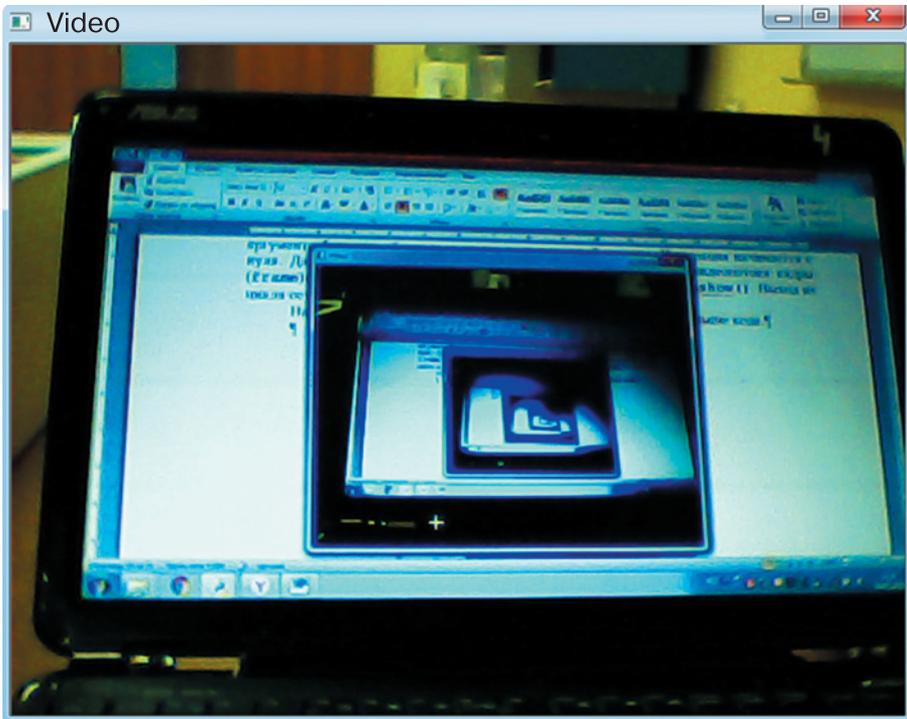


Рис. 2.21. Захват видео с web-камеры с помощью OpenCV

¹ Таблица поиска ASCII. / URL: <https://coding.tools/ru/ascii-table>

Отображение видеоданных из файла

Теперь перейдем к отображению видео из файла. Для нашего случая подойдет любой небольшой видеофайл, полученный с помощью смартфона или цифровой камеры. Однако отметим, что OpenCV не со всеми видеофайлами работает одинаково. Если ваше устройство не позволяет сохранить файл в формате AVI, можно сконвертировать его с помощью любого онлайн-сервиса, например этого¹. Не забудьте сохранить сконвертированный файл AVI в той же папке, что и файл с программой! В нашем случае это файл `video.avi`.

Итак, для того чтобы открыть видеофайл, нам потребуется внести небольшие изменения в предыдущий программный код. В результате получим следующее:

```
import cv2
# связываем файл video.avi с переменной capImg
capImg = cv2.VideoCapture('video.avi')
# открываем в цикле файл
while (capImg.isOpened()):
# получаем кадр из видеопотока файла
# кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
# если кадры закончились
    if frame is None:
        break # прерываем работу цикла
# показываем кадр из файла в окне video_file
    cv2.imshow("video_file", frame)
# организуем выход из цикла по нажатию клавиши,
# ждем 30 миллисекунд нажатия, записываем код
# нажатой клавиши
    key_press = cv2.waitKey(30)
# если код нажатой клавиши совпадает с кодом «q»,
    if key_press == ord('q'):
        break # то прервать цикл while
# освобождаем память от переменной capImg
capImg.release()
# закрываем все окна opencv
cv2.destroyAllWindows()
```

¹ Онлайн-конвертер видео. URL: <https://fconvert.ru/video/>

Как видно из программного кода, для открытия видеофайла используется та же функция `cv2.VideoCapture()`. Однако в качестве аргумента функции используется непосредственно имя видеофайла или полный путь до него, если видеофайл находится в другой папке. Далее, так же как и с web-камерой, считываем покадрово данные из видеопотока. Если кадры закончились, прерываем работу цикла. Остальная часть программы аналогична предыдущему примеру. На **рис. 2.22** показано окно для покадрового воспроизведения видеофайла.

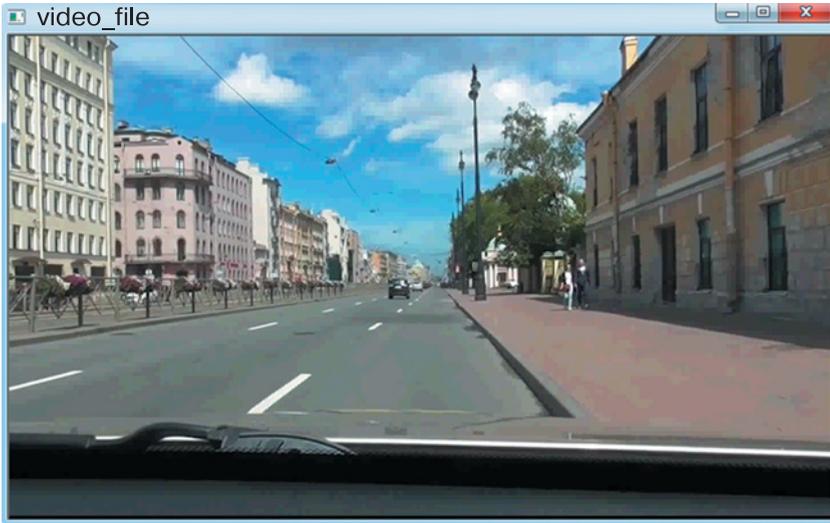


Рис. 2.22. Воспроизведение файла `video.avi` с помощью OpenCV (Источник: кадр из ролика https://www.youtube.com/watch?time_continue=2&v=eC6KtIYOlf4&feature=emb_logo)

ЗАДАНИЯ

1. Измените размер отображаемого кадра, используя функцию `cv2.resize()`.
2. Выполните зеркальное отражение кадра с помощью функции `cv2.flip()`.

Поиск цветных объектов на видео с помощью OpenCV-Python

Поиск цветного объекта в кадрах видеопотока является базовой задачей в системах с компьютерным зрением. Подобные задачи решаются при разработке автономных систем управления беспилотных автомобилей. Это идентификация дорожных знаков, сигналов светофора, линий дорожной разметки и др. Попробуем разобраться, как решаются такие задачи.

Мы с вами уже находили цветные объекты на фотографиях. При этом, кроме самого объекта, обнаруживалось большое количество мелких крапинок, которые не являются искомым объектом. Чтобы таких крапинок было поменьше, мы исключим из поиска те области, где искомый объект визуально не присутствует, то есть поиск будем проводить внутри границ некоторой области кадра. Для этого предварительно просмотрим видео и определим границы и размеры области, в которой присутствуют интересующие нас искомые объекты. В нашем примере это изображения дорожных знаков. Приблизительную оценку расположения и размеров нашей области поиска можно сделать с учетом того, что левый верхний угол кадра имеет координаты $(0, 0)$, а правый нижний — это значения размеров кадра. Окончательные размеры области подберем опытным путем. Принимая за основу программу, описанную в пункте «Поиск объекта в цветовом HSV-пространстве», составим порядок действий для поиска цветного объекта в видеопотоке.

1. Получить доступ к графическим данным видеопотока: открытие видеофайла или захват с видеокамеры.
2. Перевести изображение кадра в цветовое HSV-пространство.
3. Провести обрезку кадра по размерам видимой области с искомым объектом.
4. Подобрать границы цветового диапазона для цветовой маски.
5. Применить цветовую HSV-маску к кадрам видеопотока, переведенным в HSV-пространство.
6. Отобразить результаты наложения цветовой HSV-маски в отдельном окне.

Составим небольшую программу, которая позволит реализовать приведенный выше алгоритм. Мы будем находить объекты синего цвета (дорожные знаки на синем фоне) в видеопотоке из файла, полученного с автомобильного видеорегистратора.

```
# подключение библиотеки numpy
# под именем np
import numpy as np
# подключение библиотеки opencv
import cv2
# связываем видеопоток файла video.avi с переменной
# capImg
capImg = cv2.VideoCapture('video.avi')
# открываем файл с видео
while(capImg.isOpened()):
# получаем кадр из видеопотока файла,
# кадры по очереди считываются в переменную frame
ret, frame = cap.read()
# если кадры закончились, то прерываем цикл
if frame is None:
    break
# переводим кадр в цветовое HSV-пространство
frame_hsv = cv2.cvtColor(frame,
                           cv2.COLOR_BGR2HSV)
# делаем вырезку области кадра, где ожидается
# объект
crop_frame = frame[60:270, 30:750]
crop_frame_hsv = frame_hsv[60:270, 30:750]
# задаем нижнюю и верхнюю границы цветового фильтра
# с помощью массивов numpy
# цвет 0...180,
# насыщенность 0 - блеклый, 255 - насыщенный
# яркость 0 - темный, 255 - светлый
low_Blue = np.array([105, 150, 0],
                    dtype = "uint8")
high_Blue = np.array([135, 255, 210],
                     dtype = "uint8")
# применяем маску по цвету к фрагменту кадра
blue_mask = cv2.inRange(crop_frame_hsv, low_Blue,
                        high_Blue)
# показываем области кадров с объектом
# в окне video_mask результат наложения маски,
cv2.imshow("video_mask", blue_mask)
# в окне video_frame вырезку из кадра
cv2.imshow("video_frame", crop_frame)
# организуем выход из цикла по нажатию клавиши
```

```
# ждем 30 миллисекунд нажатия, записываем код
# нажатой клавиши
key_press = cv2.waitKey(30)
# если код нажатой клавиши совпадает с кодом «q»,
if key_press == ord('q'):
    break
# освобождаем память от переменной capImg
capImg.release()
# закрываем все окна opencv
cv2.destroyAllWindows()
```

Результат работы программы показан на рис. 2.23.

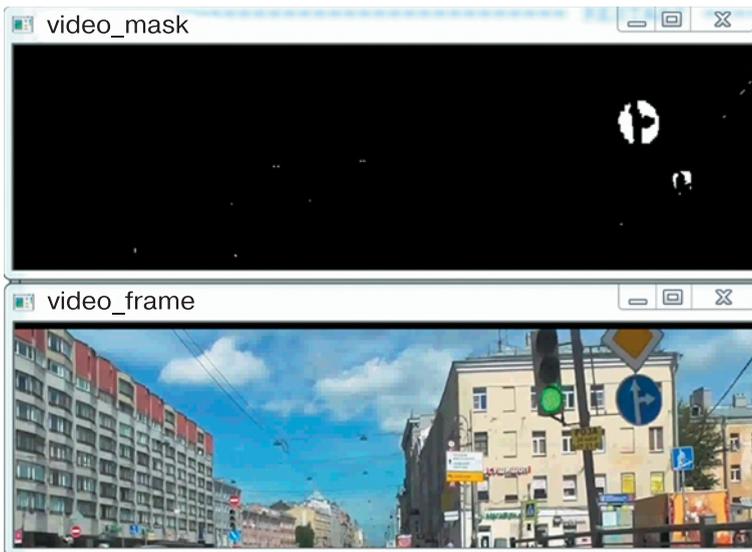


Рис. 2.23. Результат поиска дорожных знаков с синим фоном в видео-файле с автомобильного видеорежистратора (Источник: кадр из ролика https://www.youtube.com/watch?time_continue=2&v=eC6KtlYOlF4&feature=emb_logo)

В этой программе в строке «`import numpy as np`» мы импортировали дополнительно пакет `numpy` под именем `np`. Этот пакет необходим, так как в OpenCV все изображения представлены как массивы `ndarray`. Далее следуют уже знакомые строки, связанные с открытием видеофайла и чтением кадров. После конвертации кадров в HSV-пространство с помощью функции `cv2.cvtColor()` выполняем обрезку кадров по границам области, в которой будет размещен искомый

объект. Параметры срезов для вашего видеофайла, возможно, будут другими. Это зависит от размеров кадра и местоположения искомого объекта. Как правильно определить параметры срезов, показано на рис. 2.9. Выделив такую область в кадре, мы значительно уменьшаем или даже исключаем нахождение ложных объектов — групп точек, не являющихся объектом. Далее подбираем нижнюю и верхнюю границы цветового фильтра **low_Blue** и **high_Blue** так, чтобы получить предельно четкое изображение объекта на черном фоне (см. рис. 2.23). Затем с помощью функции **cv2.imshow()** выводим на экран в отдельных окнах области кадра с объектом.

ЗАДАНИЕ

Подберите границы цветовых диапазонов для поиска объектов других цветов (красный, желтый, зеленый и т. д.) на видео.

Суммирование цветовых масок

Итак, мы изучили, как осуществляется поиск единичного объекта, находящегося в поле кадра видеопотока. Однако современные системы компьютерного зрения могут обнаруживать множество различных объектов, попавших в поле зрения видеокамеры. Рассмотрим задачу поиска нескольких объектов, различающихся по цветам.

Чтобы найти группу объектов, различающихся по цветам, необходимо определить цветовую маску каждого объекта. Затем полученные маски сложить и наложить их на исходное изображение. В результате мы получим составной цветовой фильтр, который позволит отобразить искомую группу объектов. Давайте разберем работу программы, которая в потоке видео находит группы объектов красного, желтого и синего цветов. Взяв за основу предыдущий программный код, составим следующую программу:

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# связываем видеопоток файла video.avi с переменной
# capImg
```

```
capImg = cv2.VideoCapture('video.avi')
# открываем файл с видео
while (capImg.isOpened()):
# получаем кадр из видеопотока файла,
# кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
# если кадры закончились, то прерываем цикл
    if frame is None:
        break
# переводим кадр в цветовое пространство HSV
    frame_hsv = cv2.cvtColor(frame,
                               cv2.COLOR_BGR2HSV)
# делаем вырезку области кадра, где ожидается
# группа объектов
    crop_frame = frame[60:270, 30:750]
    crop_frame_hsv = frame_hsv[60:270, 30:750]
# задаем нижние и верхние границы цветовых фильтров
# с помощью массивов numpy
# цвет 0...180
# насыщенность 0 - блеклый, 255 - насыщенный
# яркость 0 - темный, 255 - светлый
# фильтр для синего цвета - подобрали ранее
    low_Blue = np.array ([105, 150, 0],
                          dtype = "uint8")
    high_Blue = np.array([135, 255, 210],
                          dtype = "uint8")
# фильтр для желтого цвета
    low_Yel = np.array([10, 150, 100],
                       dtype = "uint8")
    high_Yel = np.array([50, 255, 255],
                        dtype = "uint8")
# фильтр для красного цвета
# красный цвет представляет собой две области,
# красный в сторону оранжевой области
    low_Red_O = np.array([0, 85, 110],
                          dtype = "uint8")
    high_Red_O = np.array([5, 165, 155],
                           dtype = "uint8")
# красный в сторону фиолетовой области
    low_Red_V = np.array([165, 55, 40],
                          dtype = "uint8")
```

```

high_Red_V = np.array([180, 105, 120],
                      dtype = "uint8")
# применяем маску по каждому цвету к фрагменту
# кадра для выделения синего цвета
blue_mask = cv2.inRange(crop_frame_hsv, low_Blue,
                       high_Blue)
# для выделения желтого цвета
yel_mask = cv2.inRange(crop_frame_hsv, low_Yel,
                      high_Yel)
# для выделения красного накладываются две маски
red1_mask = cv2.inRange(crop_frame_hsv,
                       low_Red_O, high_Red_O)
red2_mask = cv2.inRange(crop_frame_hsv,
                       low_Red_V, high_Red_V)
# вычисляем полную маску
# полная маска представляет собой сумму всех масок
full_mask = red1_mask + red2_mask + blue_mask +
            yel_mask
# показываем области кадров с искомой группой
# объектов: в окне video_mask – результат наложения
# полной маски,
cv2.imshow("video_mask", full_mask)
# в окне video_frame – вырезку из кадра
cv2.imshow("video_frame", crop_frame)
# организуем выход из цикла по нажатию клавиши,
# ждем 30 миллисекунд нажатия, записываем код
# нажатой клавиши
key_press = cv2.waitKey(30)
# если код нажатой клавиши совпадает с кодом «q»,
if key_press == ord('q'):
    break
# освобождаем память от переменной capImg
capImg.release()
# закрываем все окна opencv
cv2.destroyAllWindows()

```

Как видно из текста программы, мы добавили несколько строк для определения границ фильтров желтого и красного цветов. Если фильтр для желтого цвета состоит из одной области с границами **low_Yel** и **high_Yel**, то для красного цвета таких областей две. Границы этих областей в программе обозначены как **low_Red_O**, **high_Red_O**

и `low_Red_V`, `high_Red_V`. Это объясняется тем, что красный цвет находится на границе шкалы `H` пространства `HSV`. Оттенки красного цвета (см. рис. 2.17) присутствуют в оранжевой и фиолетовой областях цветового пространства. Поэтому фильтр для красного будет состоять из двух областей: в начале шкалы `H` — красно-оранжевая область и в конце шкалы `H` — фиолетово-красная область. Сложив все области, мы получим составной фильтр, условные границы областей которого изображены на цветовой диаграмме (рис. 2.24). Не забываем, что в `OpenCV` шкала для `H` кодируется от 0 до 180, а не до 360, как изображено на рисунке. Соответственно все значения параметра цвета из данного рисунка уменьшаются в два раза!

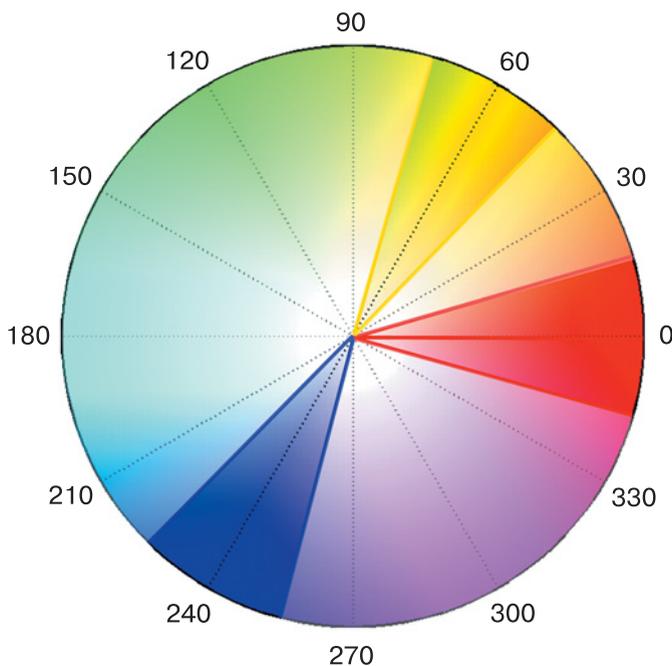


Рис. 2.24. Цветовая диаграмма `HSV`: границы областей составного цветового фильтра

После того как были заданы все границы цветowych фильтров, получаем с помощью функции `cv2.inRange()` маску по каждому цвету применительно к нашему фрагменту кадра `crop_frame_hsv`. Результаты наложения масок (переменные `blue_mask`, `yel_mask`, `red1_mask`, `red2_mask`) суммируются и записываются в переменную `full_mask`. Отображая с помощью функции `cv2.imshow()` на экране полную маску, получим искомую группу объектов.

Результаты работы данной программы представлены на **рис. 2.25**. Прямоугольником обведена именно та группа объектов, на которую был нацелен поиск. Как видно из рисунка, при суммировании масок появляются также и ложные артефакты. Обратите внимание на группу белых пятен слева. Размеры некоторых пятен сопоставимы с размерами искомых объектов. А это может привести к определенным трудностям при идентификации нужных объектов. Чтобы избежать подобных проблем на данном этапе, необходимо осуществлять более локализованный поиск, то есть только в той области кадра, где действительно находится искомый объект или группа объектов. Для этого требуется выполнять срезы более тщательно.

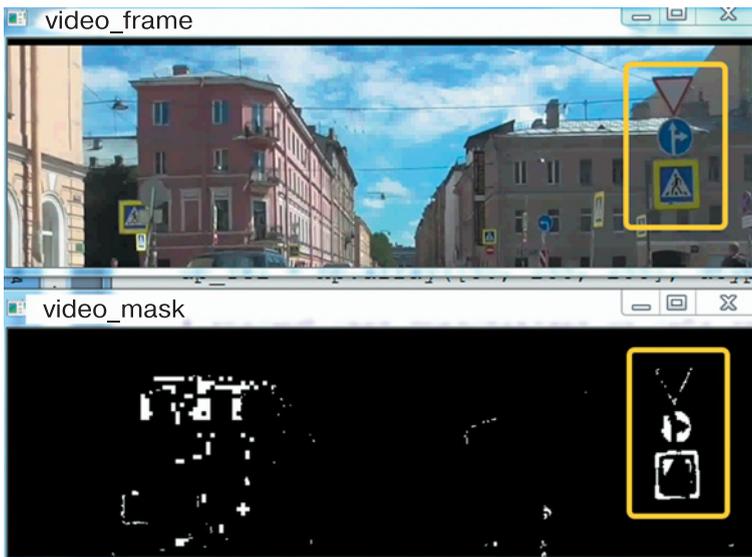


Рис. 2.25. Результат поиска нескольких дорожных знаков на данных с автомобильного видеорежистратора (Источник: вырезка кадра из ролика https://www.youtube.com/watch?time_continue=2&v=eC6KtlYOIf4&feature=emb_logo)

ЗАДАНИЯ

1. Проверьте работу каждого цветового фильтра в отдельном окне, отображая соответствующий видеоряд с помощью команды `cv2.imshow(«video_mask», <маска_по_цвету>)`

2. Подключите к компьютеру web-камеру. Внесите изменения в приведенный выше программный код для работы с видеопотоком с web-камеры. Проверьте работу составного фильтра, используя в качестве объектов, например, цветные кирпичики или балки от Lego. При необходимости уточните значения нижних и верхних границ цветовых фильтров.

Настройка цветового фильтра средствами OpenCV

Подбирать цветовой фильтр для поиска того или иного объекта, каждый раз запуская программу с новыми значениями, не очень удобно. В OpenCV есть простейшие инструменты, которые позволяют в режиме реального времени настраивать все три компонента HSV для подбора цветового фильтра. Попробуем создать для себя несложный графический интерфейс по настройке цветового фильтра. Интерфейс будет представлять собой отдельное окно, в котором размещены ползунки, задающие диапазоны для компонент H, S и V. Меняя положение ползунков, будем задавать диапазоны по каждой компоненте пространства HSV и визуально оценивать результат фильтрации. Для этого нам потребуются три функции:

- 1) `cv2.namedWindow()` — создать окно;
- 2) `cv2.createTrackbar()` — создать графический компонент в виде ползунка с указателем;
- 3) `cv2.getTrackbarPos()` — считать положение указателя ползунка.

При создании окна для функции `cv2.namedWindow()` нужно указать один или два аргумента в следующем порядке:

```
cv2.namedWindow(<имя_окна>[, flag])
```

где аргумент `<имя_окна>` означает название окна (заголовка), которое может быть использовано в качестве идентификатора окна, причем текстовое значение заключается в апострофах (' ') или в кавычках (" "); аргумент `[, flag]` — флаг окна — необязательный параметр, который определяет свойства и вид окна.

Поддерживаются следующие флаги:

- `cv2.WINDOW_NORMAL` — можно изменять размер окна (без ограничений);

- **cv2.WINDOW_AUTOSIZE** — размер окна настраивается автоматически в соответствии с отображаемым изображением, в этом случае размер окна вручную изменить нельзя;
- **cv2.WINDOW_KEEPRATIO** — размер окна настраивается с учетом пропорции изображения;
- **cv2.WINDOW_FREERATIO** — размер окна настраивается без учета пропорции изображения.

Если флаг не указан явно, то система по умолчанию применяет значение **cv2.WINDOW_AUTOSIZE**. Поскольку мы создаем окно настроек, флаг можно явно не указывать. Остается один аргумент — это название окна, и его нам придется придумать.

Для создания графического компонента «ползунок с указателем» в определении функции **cv2.createTrackbar()** необходимо указать следующие аргументы:

```
cv2.createTrackbar(trackbarName, windowName, value, count, onChange)
```

где аргументы означают следующее:

- **trackbarName** — название ползунка, текстовое значение заключается в апострофах (' ') или в кавычках (" ");
- **windowName** — название окна, в котором создается компонент, текстовое значение заключается в апострофах (' ') или в кавычках (" ");
- **value** — целочисленное значение, которое отражает положение указателя ползунка;
- **count** — максимальное значение указателя ползунка, минимальное значение всегда равно 0;
- **onChange** — указатель на функцию-обработчик, вызываемую при каждом изменении указателя ползунка.

Таких ползунков у нас будет шесть, по два на каждую компоненту: один — для регулировки нижней границы, другой — для регулировки верхней границы. Рекомендуется подобрать ползункам названия, соответствующие по смыслу компонентам HSV, например: H1, H2, S1, S2, V1, V2 или Hlo, Hhi, Slo, Shi, Vlo, Vhi и т. д. Ползунки, отвечающие за нижнюю границу диапазонов, должны иметь начальное положение указателя, равное нулю. Начальное положение указателей ползунков верхней границы должно быть равно максимальному значению шкалы — **count**. Напоминаем, что для компоненты H это

равно 180, а для компонент S и V равно 255. В качестве обработчика создадим «пустую» функцию, которая ничего не будет делать, так как для нашего случая достаточно просто считать текущие значения указателей ползунков.

Для считывания указателей ползунков мы воспользуемся функцией `cv2.getTrackbarPos()`, в определении которой надо указать два аргумента:

```
cv2.getTrackbarPos(trackbarName, windowName),
```

где аргументы означают следующее:

- **trackbarName** — название ползунка, текстовое значение заключается в апострофах (' ') или в кавычках (" ");
- **windowName** — название окна, в котором находится данный компонент, текстовое значение заключается в апострофах (' ') или в кавычках (" ").

Функция `cv2.getTrackbarPos()` возвращает целочисленное значение от 0 до величины **count**, указанной при создании текущего ползунка.

Рассмотрим работу нашего простейшего интерфейса на примере следующего программного кода:

```
# подключение библиотеки numpy  
import numpy as np  
# подключение библиотеки opencv  
import cv2  
  
# создаем «пустую» функцию  
def nothing(*arg):  
    pass  
  
# создаем окно настроек с именем Set  
cv2.namedWindow('Set')  
# создаем 6 ползунков для настройки цветового фильтра  
# для компонент H,S,V соответственно  
cv2.createTrackbar('h1', 'Set', 0, 180, nothing)  
cv2.createTrackbar('s1', 'Set', 0, 255, nothing)  
cv2.createTrackbar('v1', 'Set', 0, 255, nothing)  
cv2.createTrackbar('h2', 'Set', 180, 180, nothing)  
cv2.createTrackbar('s2', 'Set', 255, 255, nothing)
```

```

cv2.createTrackbar('v2', 'Set', 255, 255, nothing)
# связываем видеопоток файла video.avi с переменной
# capImg
capImg = cv2.VideoCapture('video.avi')
# открываем файл с видео
while(capImg.isOpened()):
# получаем кадр из видеопотока файла,
# кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
# если кадры закончились, то прерываем цикл
    if frame is None:
        break
# переводим кадр в цветовое пространство HSV
    frame_hsv = cv2.cvtColor(frame,
                               cv2.COLOR_BGR2HSV )
# вводим логическую переменную для выхода из
# программы
    cl_pr = False
# запускаем цикл со стоп-кадром по настройке
# фильтра
    while True:
#         считываем значения бегунков для H,S,V
#         соответственно
        h1 = cv2.getTrackbarPos('h1', 'Set')
        s1 = cv2.getTrackbarPos('s1', 'Set')
        v1 = cv2.getTrackbarPos('v1', 'Set')
        h2 = cv2.getTrackbarPos('h2', 'Set')
        s2 = cv2.getTrackbarPos('s2', 'Set')
        v2 = cv2.getTrackbarPos('v2', 'Set')
#         формируем нижнюю и верхнюю границы цветового
#         фильтра
        h_min = np.array((h1, s1, v1), np.uint8)
        h_max = np.array((h2, s2, v2), np.uint8)
#         накладываем фильтр на кадр в цветовом
#         пространстве HSV,
#         результат записываем в RealTimeMask
        RealTimeMask = cv2.inRange(frame_hsv, h_min,
                                    h_max)
#         показываем исходный кадр в окне Original
        cv2.imshow('Original', frame)
#         показываем фильтрованный кадр в окне result

```

```

cv2.imshow('result', RealTimeMask)
#    организуем управление программой по нажатию
#    клавиши, ждем 30 миллисекунд нажатия,
#    записываем код нажатой клавиши
key_press = cv2.waitKey(30)
#    если код клавиши равен коду символа «n»
if key_press == ord('n'):
#    прерываем цикл показа текущего кадра
#    для перехода к следующему
    break
#    если код клавиши равен коду символа «q»
elif key_press == ord('q'):
#    переменной выхода из программы присваиваем
#    True
    cl_pr = True
#    прерываем цикл показа текущего кадра
    break
#    если переменная выхода из программы True
if cl_pr:
#    прерываем цикл считывания кадров
    break
capImg.release()
cv2.destroyAllWindows()

```

Программа работает следующим образом. После импортирования необходимых модулей (**numpy** и **OpenCV**) следует объявление функции обработчика — **nothing**. Как мы уже сказали, в нашем случае функция-обработчик не должна ничего делать. Единственная строка нашей функции содержит оператор **pass**, так как в Python нельзя объявлять функции с пустыми строками. Обычно оператор **pass** используют как оператор-заглушку там, где код планируется, но он еще не написан. Данный оператор ничего не выполняет. Символ «*» в определении аргументов функции ставится для того, чтобы функция могла принимать переменное количество позиционных аргументов.

Основной код программы начинается с создания интерфейсного окна. Эта часть реализуется с помощью функций **cv2.namedWindow()** и **cv2.createTrackbar**. Созданное нами окно имеет заголовок **Set** и содержит шесть ползунков для настройки диапазонов цветового фильтра HSV. Затем следует уже знакомый фрагмент кода для открытия видеофайла, а также считывания и перевода кадров в цве-

товое пространство HSV. Далее в цикле «**while True:**» выполняется покадровое отображение видеопотока в режиме «стоп-кадр». Внутри данного цикла считываются значения указателей ползунков, по которым определяется нижняя и верхняя границы цветового фильтра. Полученный фильтр применяется к отображаемому кадру, а результат мгновенно выводится в окне **result** с помощью функции **cv2.imshow()**. Нажимая клавишу «n», мы переходим к обработке следующего кадра. Чтобы закончить работу программы, следует нажать клавишу «q».

Таким образом, мы получили эффективный инструмент для настройки цифровых фильтров в пространстве HSV. Используя интерфейсное окно с настройками, можно в процессе прокрутки видео подобрать параметры цветовой маски для фильтрации объектов нужного цвета, визуальнo оценивая результат фильтрации. На **рис. 2.26**



Рис. 2.26. Результат поиска нескольких дорожных знаков в данных с автомобильного видеорежистратора (Источник: кадр из ролика https://www.youtube.com/watch?time_continue=2&v=eC6KtIYOIf4&feature=emb_logo)

показано окно **Set**, в котором с помощью шести ползунков легко и быстро был подобран фильтр для объектов синего цвета. Результат наложения фильтра изображен в окне **result**. Аналогичным образом подбираются фильтры для других цветов.

ЗАДАНИЕ

Подключите к компьютеру web-камеру. Опираясь на предыдущее задание, внесите изменения в приведенный выше программный код для работы с видеопотоком с web-камеры. Проверьте работу интерфейса для настройки цветового фильтра, используя различные цветные объекты.

Поиск цветных объектов на видео в реальном времени

Имея в своем распоряжении эффективный инструмент для настройки цветных фильтров, попробуем улучшить результат поиска цветного объекта в видеопотоке. Все необходимые для этого функции мы уже изучили. Тестировать программу будем на записи с автомобильного видеорегистратора. В качестве объектов для поиска определим изображения дорожных знаков, попавших в поле зрения объектива камеры видеорегистратора.

Чтобы в работе максимально исключить поиск «ложных» цветных объектов, сделаем обрезку кадра по границам искомого объекта. Затем вырезанную область увеличим в два-три раза, коэффициент можно будет подобрать позднее, в процессе отладки. Это необходимо для увеличения площади изображения самих объектов и более точного определения их координат. Координаты будем находить с помощью функции моментов **cv2.moments()**. Красным маркером пометим точку с полученными координатами.

Пролистывание кадров с видеозаписи, как и в предыдущем коде, будем осуществлять нажатием клавиши «n». С помощью окна настройки цветного фильтра **Set** произведем визуальный поиск нужного нам цветного объекта. Завершение работы программы осуществляется нажатием клавиши «q». Ниже приводится код программы.

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2

# создаем «пустую» функцию маски
def nothing(*arg):
    pass

# создаем окно настроек с именем Set
cv2.namedWindow('Set')
# создаем 6 ползунков для настройки цветового фильтра
# для компонент H,S,V соответственно
cv2.createTrackbar('h1', 'Set', 0, 180, nothing)
cv2.createTrackbar('s1', 'Set', 0, 255, nothing)
cv2.createTrackbar('v1', 'Set', 0, 255, nothing)
cv2.createTrackbar('h2', 'Set', 180, 180, nothing)
cv2.createTrackbar('s2', 'Set', 255, 255, nothing)
cv2.createTrackbar('v2', 'Set', 255, 255, nothing)
# связываем видеопоток файла video.avi с переменной
# capImg
capImg = cv2.VideoCapture('video.avi')
# открываем файл с видео
while (capImg.isOpened()):
    # получаем кадр из видеопотока файла
    # кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
    # если кадры закончились, то прерываем цикл
    if frame is None:
        break
    # вырезаем область кадра, где ожидаются знаки,
    # результат записываем в cr_frame
    cr_frame = frame[150:260, 485:600]
    # задаем коэффициент увеличения k
    k = 2
    # рассчитываем новые размеры для вырезанного
    # фрагмента
    x = cr_frame.shape[1] * k
    y = cr_frame.shape[0] * k
    dim = (x, y)
    # увеличиваем размер фрагмента кадра,
```

```
# результат записываем в RSframe
RSframe = cv2.resize(cr_frame, dim,
                    interpolation = cv2.INTER_AREA)
# переводим трансформированный фрагмент в HSV,
# результат записываем в RSframeHSV
RSframeHSV = cv2.cvtColor(RSframe,
                          cv2.COLOR_BGR2HSV)
# вводим логическую переменную для выхода из
# программы
cl_pr = False
# запускаем цикл со стоп-кадром по настройке
# фильтра
while True:
#     считываем значения ползунков для H,S,V
#     соответственно
h1 = cv2.getTrackbarPos('h1', 'Set')
s1 = cv2.getTrackbarPos('s1', 'Set')
v1 = cv2.getTrackbarPos('v1', 'Set')
h2 = cv2.getTrackbarPos('h2', 'Set')
s2 = cv2.getTrackbarPos('s2', 'Set')
v2 = cv2.getTrackbarPos('v2', 'Set')
#     формируем нижнюю и верхнюю границы цветового
#     фильтра
h_min = np.array((h1, s1, v1), np.uint8)
h_max = np.array((h2, s2, v2), np.uint8)
#     накладываем фильтр на кадр в цветовом
#     пространстве HSV, результат записываем
#     в RealTimeMask
RealTimeMask = cv2.inRange(RSframeHSV, h_min,
                          h_max)
#     вычисляем моменты отфильтрованного
#     HSV-изображения
moments = cv2.moments(RealTimeMask, 1)
#     вычисляем сумму X, Y координат всех точек
#     пятна-объекта
dm01 = moments['m01']
dm10 = moments['m10']
Area = moments['m00']
#     вычисляем средние координаты центра
#     пятна-объекта
```

```

x = int(dm10 / Area)
y = int(dm01 / Area)
# ставим красную метку по полученным
# координатам
cv2.circle(RSframe, (x, y), 10, (0,0,255),
          -1)
# показываем кадры в opencv-окне
cv2.imshow("video_frame", frame)
cv2.imshow("Crop_frame", RSframe)
cv2.imshow("video_mask", RealTimeMask)
# организуем управление программой по нажатию
# клавиши, ждем 30 миллисекунд нажатия,
# записываем код нажатой клавиши
key_press = cv2.waitKey(30)
# если код клавиши равен коду символа «n»
if key_press == ord('n'):
# прерываем цикл показа текущего кадра
# для перехода к следующему
    break
# если код клавиши равен коду символа «q»
elif key_press == ord('q'):
# переменной выхода из программы присваиваем
# True
    cl_pr = True
# прерываем цикл показа текущего кадра
    break
# если переменная выхода из программы True
if cl_pr:
# прерываем цикл считывания кадров
    break
capImg.release()
cv2.destroyAllWindows()

```

На рис. 2.27. изображен результат поиска дорожных знаков по синему (1) и желтому (2) фильтрам. Красным маркером указан центр найденного объекта. Как видно из рисунка, поиск в более компактной вырезанной области заметно уменьшает проявление «ложных» элементов, не относящихся к исходному объекту. Таким образом, тщательно выполняя вырезку из кадра фрагмента с объектом, с помощью визуального настраиваемого фильтра можно более качественно произвести поиск цветного объекта.

Здесь следует сделать небольшое примечание. В наших примерах по поиску объектов в этом разделе и ранее рассмотрены сильно упрощенные постановки задач. В реальной практике чаще всего заранее неизвестно, в какой области кадра может появиться искомым объект. Решение таких задач предполагает разработку специальных программ с использованием нейронных сетей для идентификации искомым объектов.

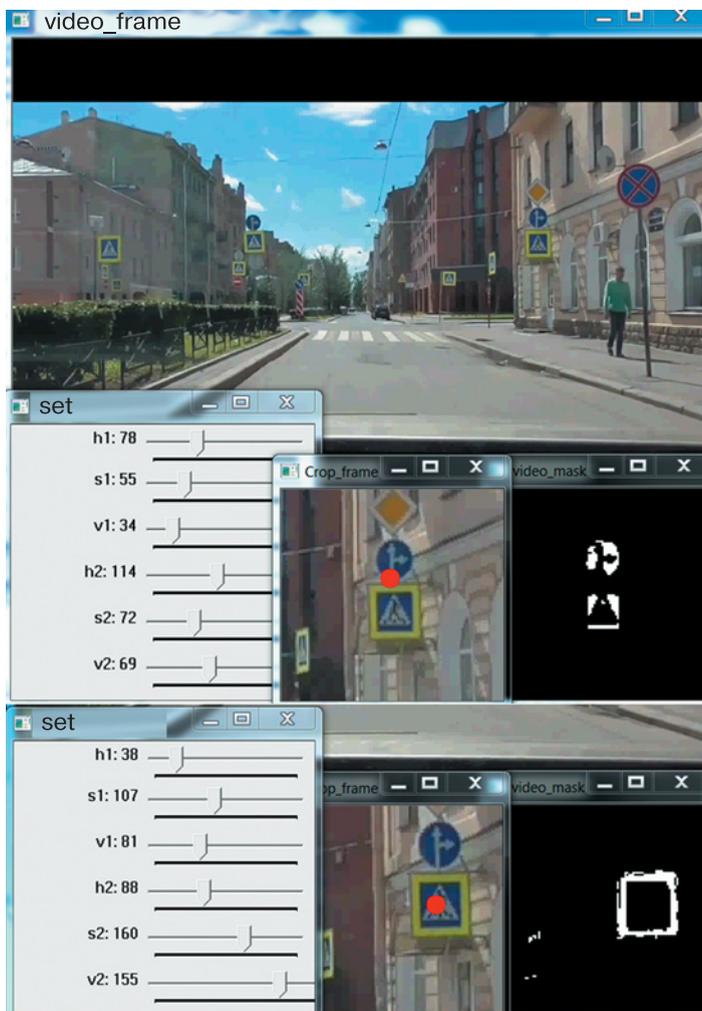


Рис. 2.27. Результат поиска дорожных знаков на вырезанной области кадра с автомобильного видеорежистратора (Источник: кадр из ролика https://www.youtube.com/watch?time_continue=2&v=eC6KtIYOIf4&feature=emb_logo)

ЗАДАНИЕ

Проведите эксперимент, увеличив значение коэффициента трансформации вырезанной области кадра, и сравните результат поиска.

Выделение контуров объектов с помощью OpenCV-Python

Получив навыки работы с цветовыми фильтрами, рассмотрим следующий немаловажный инструмент машинного зрения — выделение **контуров объектов**. Под контуром объекта понимается его видимый край, который разграничивает объект и фон. В большинстве методов анализ изображений состоит в работе с контурами, а не с пикселями. Совокупность методов выделения, описания и преобразования контуров изображения называется **контурным анализом**. Каждый контур хранится в виде двумерного массива библиотеки **numpy**. Это еще один объект **numpy**, с которым мы будем работать при изучении компьютерного зрения.

Для поиска контуров в библиотеке OpenCV имеется функция **cv2.findContours()**, которая возвращает два параметра: **contours** и **hierarchy**¹ (для версии OpenCV 4.x.x). В общем виде функция имеет следующее представление:

```
cv2.findContours (image, mode, method[,  
                 contours[, hierarchy[, offset]])
```

где аргументы означают следующее:

- **image** — переменная, в которую записано графическое изображение, специально подготовленное для анализа. В данном случае для работы используется 8-битное монохромное изображение.
- **mode** — режим группировки найденных контуров для управления их отображением. Таких режимов четыре:

¹ В OpenCV версии 3.x.x и более ранних возвращается три параметра: `image`, `contours`, `hierarchy`. В версии с интерфейсом `cv` возвращается только параметр `contours`. Подробнее см. документацию по OpenCV. URL: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html (Дата обращения 20.12.2019).

- 1) **cv2.RETR_LIST** — режим выдает все найденные контуры без группировки;
 - 2) **cv2.RETR_EXTERNAL** — режим выдает только крайние внешние контуры, а контуры, найденные внутри объекта, не отображаются;
 - 3) **cv2.RETR_CCMP** — режим группировки контуров в двухуровневую иерархию:
 - на верхнем уровне — внешние контуры объекта,
 - на втором уровне — контуры имеющихся отверстий,
 - все остальные контуры группируются на верхнем уровне;
 - 4) **cv2.RETR_TREE** — режим группировки контуров в многоуровневую иерархию.
- **method** — способ хранения найденных контуров; выбирается один из трех методов упаковки контуров:
 - 1) **cv2.CHAIN_APPROX_NONE** — упаковка не применяется, хранит абсолютно все точки контура, при этом любые две последующие точки контура являются соседями по горизонтали, вертикали или диагонали со смещением по координатам не больше 1;
 - 2) **cv2.CHAIN_APPROX_SIMPLE** — метод склейки всех горизонтальных, вертикальных и диагональных сегментов контуров с сохранением только их конечных точек, например для хранения вертикального прямоугольного контура будет использовано всего четыре точки;
 - 3) **cv2.CHAIN_APPROX_TC89_L1**, **cv2.CHAIN_APPROX_TC89_KCOS** — для хранения контуров применяется метод упаковки Teh-Chin¹.
 - **contours** — (необязательный параметр) список всех обнаруженных контуров, представленных в виде векторов.
 - **hierarchy** — (необязательный параметр) информация о топологии контуров: каждый элемент иерархии представляет собой сборку из четырех индексов, соответствующую **contours [i]**:
 - 1) **hierarchy [i] [0]** — индекс следующего контура на текущем слое;
 - 2) **hierarchy [i] [1]** — индекс предыдущего контура на текущем слое;
 - 3) **hierarchy [i] [2]** — индекс первого контура на вложенном слое;
 - 4) **hierarchy [i] [3]** — индекс родительского контура.

¹ Teh, C. H. and Chin, R. T., On the Detection of Dominant Points on Digital Curve. PAMI 11 8, pp 859–872 (1989).

- **offset** — (необязательный параметр) величина дополнительного смещения, на которое смещается каждая точка контура. Этот параметр полезно применять, если контуры извлекаются из ROI-изображения, а затем должны быть проанализированы во всем контексте изображения.

ПОЯСНЕНИЕ

Region Of Interest (ROI) — представляет собой определенную пользователем прямоугольную область на рисунке (Интересующая область рисунка — ИОР), которую можно выделить для дальнейшей обработки.

После определения ИОР большинство функций OpenCV будет применяться к этой области. Это бывает полезно, когда нужно выполнить сравнения ИОР с каким-нибудь шаблоном (например, при распознавании объектов). ИОР должна обязательно находиться внутри исходного изображения.

После операции поиска контуров все найденные контуры необходимо отобразить. Визуализация найденных в кадре контуров выполняется с помощью функции `cv2.drawContours()`. Эта функция в большей степени необходима пользователю, поскольку позволяет лучше понять, как выглядят найденные контуры. В общем виде функция `cv2.drawContours()` записывается следующим образом:

```
cv2.drawContours(image, contours,  
                 contoursIdx, color[, thickness  
                 [, lineType[, hierarchy[,  
                 maxLevel[, offset]]]])
```

где аргументы означают следующее:

- **image** — переменная с изображением, поверх которого будут нарисованы контуры.
- **contours** — переменная, в которую записаны контуры, найденные функцией `cv2.findContours()`.
- **contoursIdx** — индекс контура, который следует отобразить; если указанный индекс равен «-1», то отображаются все контуры.
- **color** — цвет линии контура, задается в виде тройки целочисленных значений (B, G, R), где каждая составляющая принимает значение от 0 до 255.

- **thickness** — (необязательный параметр) толщина линии контура, задается целочисленным параметром; если задается отрицательное значение, например **thickness=CV_FILLED**, то будут рисоваться только внутренние (вложенные) контуры.

- **lineType** — (необязательный параметр) тип контурной линии; в документации OpenCV указываются три значения¹:

- 1) **8** (или не указан) — для рисования линии применяется 8-связный алгоритм Бресенхэма²;
- 2) **4** — для рисования линии применяется 4-связный алгоритм Бресенхэма³;
- 3) **cv2.LINE_AA** — рисуется сглаженная линия с применением гауссовой фильтрации.

- **hierarchy** — (необязательный параметр) информация об иерархии контуров, используется в случае, когда требуется нарисовать только некоторые контуры;

- **maxLevel** — (необязательный параметр) индекс слоя, который следует отображать:

- 1) **0** — будет отображен только выбранный контур;
- 2) **1** — будут отображены выбранный контур и все его дочерние контуры;
- 3) **2** — будут отображены выбранный контур, все его дочерние контуры, дочерние контуры дочерних контуров и т. д.

- **offset** — (необязательный параметр) величина смещения точек контура, задается парой целочисленных значений (dx , dy).

Составим небольшую программу, которая будет загружать графическое изображение в формате **jpeg**, а затем выполнит поиск и отображение найденных контуров. Для проверки работы программы нам понадобится графический файл с тестовым изображением. В качестве тестового изображения подготовим рисунок в графическом редакторе Paint, например, как на **рис. 2.28**, и сохраним его в формате JPEG под именем **image.jpg**. Для удобства работы с тестовыми изображениями (а мы их подготовим не одно!) постарайтесь, чтобы их размер был в пределах 800×600 точек. В противном случае, выводимое окно с картинкой закроет весь экран.

¹ OpenCV 2.4.13.7 documentation. Drawing functions // URL: https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html (Дата обращения 18.12.2019).

^{2, 3} Турлапов В. Е. Базовые растровые алгоритмы. Растеризация линий. Алгоритмы закрашивания // URL: http://www.graph.unn.ru/rus/materials/CG/CG07_RasterAlgorithms.pdf (Дата обращения 18.12.2019).



Рис. 2.28. Пример графического изображения для поиска и отображения контуров

Наша программа будет работать по следующему алгоритму:

1. Получить доступ к данным графического файла.
2. Перевести изображение в цветное HSV-пространство.
3. Выделить объект с помощью цветового HSV-фильтра.
4. Выполнить поиск всех контуров.
5. Отобразить все найденные контуры на исходном изображении.
6. Вывести на экран окно с исходным изображением.

Рассмотрим программную реализацию этого алгоритма на нижеприведенном коде:

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# задаем путь к файлу с картинкой
# если файл с картинкой и файл с программой находятся
# в одной папке, то это будет просто имя графического
# файла
fileName = 'image.jpg'
# считываем данные графического файла в переменную
# image
image = cv2.imread(fileName)
# конвертируем исходное изображение в цветовую модель
# HSV
# результат записываем в переменную hsv_img
hsv_img = cv2.cvtColor( image, cv2.COLOR_BGR2HSV )
```

```
# подбираем параметры цветового фильтра для выделения
# нашего объекта (указанные числовые значения могут
# отличаться)
hsv_min = np.array((2, 28, 65), np.uint8)
hsv_max = np.array((26, 238, 255), np.uint8)
# применяем цветовой фильтр к исходному изображению,
# результат записываем в переменную hsv_msk
hsv_msk = cv2.inRange( hsv_img, hsv_min, hsv_max )
# ищем контуры и записываем их в переменную contours
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours( hsv_msk,
                                       cv2.RETR_LIST,
                                       cv2.CHAIN_APPROX_SIMPLE)
# отображаем все контуры поверх исходного изображения,
# цвет синий, толщина линии 3, сглаженная
cv2.drawContours( image, contours, -1, (255,0,0),
                 3, cv2.LINE_AA, hierarchy, 2)
# выводим итоговое изображение в окно contours
cv2.imshow('contours', image)
# ждем нажатия любой клавиши и закрываем все окна
cv2.waitKey()
cv2.destroyAllWindows()
```

В результате работы программы на экран будет выведено окно `contours` с исходным изображением и контурами синего цвета (рис. 2.29). Найденные контуры будут обрамлять изображенный объект на тестовой картинке.

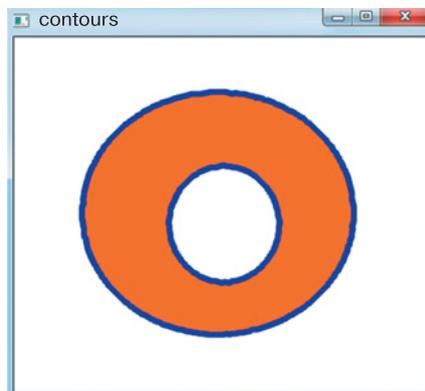


Рис. 2.29. Результат работы программы поиска и отображения контуров в тестовом изображении

ЗАДАНИЯ

1. Меняя в функции `cv2.drawContours()` параметры `contoursIdx`, `color`, `thickness` (3, 4 и 5-й аргументы по счету соответственно), выведите на экран по отдельности внешний и внутренний контуры, измените цвет контура и толщину линий контура.
2. Подготовьте с помощью редактора Paint изображение с несколькими объектами, окрашивая их в различные цвета. Подбирая параметры цветового фильтра, выведите на экран поочередно все контуры каждого объекта.

Выделение прямоугольных и эллиптических контуров

Следующим шагом в поиске контуров объектов на картинке является выделение контуров правильной геометрической формы. Здесь мы рассмотрим выделение прямоугольных и эллиптических контуров. Эта задача является более сложной. Кроме того, попутно изучим фильтрацию прямоугольных и эллиптических контуров, а также определение угла наклона прямоугольных контуров. Фильтрация контуров позволяет отсеять из анализа контуры «ложных объектов», которые программа будет находить при обработке реальных картинок, что мы уже не раз наблюдали (см. рис. 2.19, 2.23, 2.25, 2.27, 2.29). А определение угла наклона прямоугольных контуров используется при программировании роботов-манипуляторов для ситуаций, когда роботу нужно взяться за предмет под правильным углом.

Выделение прямоугольных контуров

Для выделения прямоугольных контуров в OpenCV применяется функция `cv2.minAreaRect()`, цель которой — найти прямоугольник минимальной площади, способный описать заданный замкнутый контур. Важный момент: функция `cv2.minAreaRect()` не определяет, является ли контур прямоугольным; она пытается найти прямоугольник оптимальным способом, учитывая также его вращение. В общем виде функция имеет следующее представление:

```
cv2.minAreaRect(contour)
```

где **contour** — это исходный контур, представляющий собой двухмерный массив **numpy**, который будет описан прямоугольником.

Функция **cv2.minAreaRect()** возвращает объект, назовем его **rect**, в виде прямоугольника со следующими вещественными параметрами¹: координатами центра прямоугольника (x, y), размерами (ширина, высота), углом поворота в градусах.

Чтобы отобразить замкнутый прямоугольный контур, необходимо знать все вершины прямоугольника. Для их построения нам потребуется функция **cv2.boxPoints()**. Данная функция возвращает массив вещественных координат четырех точек — вершин прямоугольника. Общий вид этой функции:

cv2.boxPoints(rect)

где **rect** — объект, возвращаемый функцией **cv2.minAreaRect()**.

Для рисования прямоугольного контура следует округлить полученные координаты до целых значений. Это можно сделать с помощью функции **numpy.int0()**, которая на входе получает вещественный массив, а возвращает целочисленный. Массив координат вершин прямоугольника, преобразованный таким образом, рисуем с помощью функции **cv2.drawContours()**.

Далее составим программу, которая все найденные на картинке контуры объекта обозначит прямоугольниками. За основу возьмем код из предыдущего параграфа «Выделение контуров объектов с помощью OpenCV-Python». Для проверки работы программы предварительно подготовим тестовое изображение с помощью редактора Paint (например, как на **рис. 2.30**), которое сохраним в папке с файлом программы под именем **rec_img.jpg**.

Поскольку за основу мы берем предыдущую программу, алгоритм будет иметь некоторое сходство, но содержать несколько иную последовательность действий:

1. Получить доступ к данным графического файла.
2. Перевести изображение в цветное HSV-пространство.
3. Выделить объект с помощью цветового HSV-фильтра.

¹ Для версии с интерфейсом **cv** (более подробно описано в источниках) возвращает структуру **Box2D**, которая содержит следующие детали: верхний левый угол (x, y), размеры (ширина, высота), угол поворота. Все параметры имеют вещественные значения, угол — в градусах. URL: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contour_features/py_contour_features.html (Дата обращения 20.12.2019).

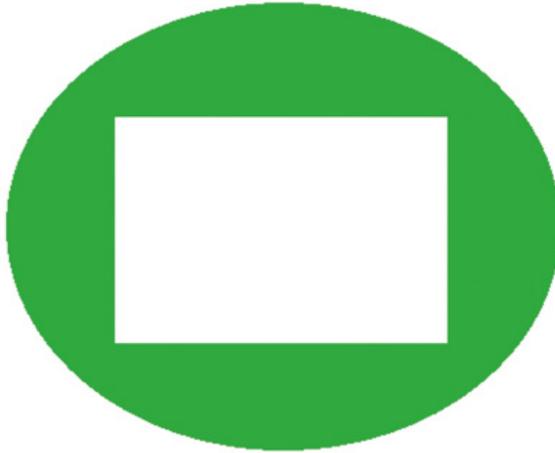


Рис. 2.30. Пример тестового изображения для поиска прямоугольных контуров

4. Выполнить поиск всех контуров.
5. Начать цикл по всем найденным контурам.
6. Найти прямоугольник, описывающий текущий контур.
7. Отобразить полученный нами прямоугольник на исходном изображении.
8. Закончить цикл.
9. Вывести на экран окно с исходным изображением.

А теперь посмотрим, что у нас получилось в программном коде:

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# задаем путь к файлу с картинкой,
# если файл с картинкой и файл с программой находятся
# в одной папке, то это будет просто имя графического
# файла
fileName = 'rec_img.jpg'
# считываем данные графического файла в переменную
# image
image = cv2.imread(fileName)
# конвертируем исходное изображение в цветовую модель
# HSV,
# результат записываем в переменную hsv_img
```

```
hsv_img = cv2.cvtColor( image, cv2.COLOR_BGR2HSV )
# подбираем параметры цветового фильтра для выделения
# нашего объекта (указанные числовые значения могут
# отличаться)
hsv_min = np.array((50, 150, 155), np.uint8)
hsv_max = np.array((70, 255, 200), np.uint8)
# применяем цветовой фильтр к исходному изображению,
# результат записываем в переменную hsv_msk
hsv_msk = cv2.inRange( hsv_img, hsv_min, hsv_max )
# ищем контуры и записываем их в переменную contours
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST, для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours( hsv_msk,
                                       cv2.RETR_LIST,
                                       cv2.CHAIN_APPROX_SIMPLE)
# перебираем все найденные контуры в цикле
for icontour in contours:
# ищем прямоугольник, результат записываем в rect
    rect = cv2.minAreaRect(icontour)
# поиск вершин прямоугольника, результат записываем
# в box
    box = cv2.boxPoints( rect )
# округление координат вершин, результат записываем
# в box
    box = np.int0( box )
# рисуем прямоугольник поверх исходного изображения
# цвет синий, толщина линии 3, поскольку рисуем
# единственный объект [box], остальные параметры
# опускаем
    cv2.drawContours( image, [box], -1, (255,0,0), 3)
# выводим итоговое изображение в окно contours
cv2.imshow('contours', image)
# ждем нажатия любой клавиши и закрываем все окна
cv2.waitKey()
cv2.destroyAllWindows()
```

В результате работы программы было выделено два прямоугольных контура (рис. 2.31). Внутренний контур полностью совпадает с прямоугольной границей объекта, а внешний прямоугольный контур описывает овальную границу объекта, причем, хотя и не совпада-

ет с ней, описывает максимально близко. Как видим, функция `cv2.minAreaRect()` построила прямоугольный контур, который максимально близко описывает непрямоугольную границу.

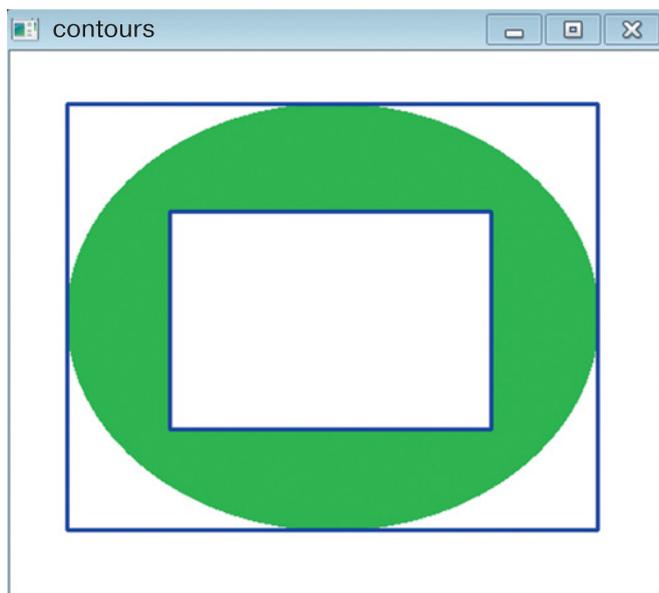


Рис. 2.31. Выделение прямоугольных контуров в тестовом изображении

ЗАДАНИЕ

Подготовьте с помощью редактора Paint изображение с несколькими объектами произвольной формы, окрашивая их в один цвет. Подобрать параметры цветового фильтра, выведите на экран прямоугольные контуры всех объектов. Измените цвет и толщину линий контуров.

Фильтрация прямоугольных контуров

Зачастую на практике при выделении контуров «нужных объектов» в силу ряда причин заодно могут быть выделены и контуры «ложных объектов» — «сор» (иначе «шум»). Как правило, это мелкие области, от которых для дальнейшего анализа необходимо избавиться. Такой

«сор» следует отфильтровать, чтобы программа при отображении контуров его игнорировала.

Решается эта задача путем вычисления площади, занимаемой каждым контуром, а затем при отображении ставится условие для контуров, площадь которых не соответствует заданному критерию. Поскольку наши контуры — это прямоугольники, то вычислить их площадь не составит труда. Для этого надо перемножить ширину и высоту. Значения ширины и высоты прямоугольников получим из функции `cv2.minAreaRect()`.

В числе параметров, возвращаемых этой функцией, есть необходимые данные. Напомним структуру объекта, возвращаемого функцией `cv2.minAreaRect()`. Это массив, содержащий **три!** элемента: **координаты** центра прямоугольника, его **размеры** (ширина и высота), **угол поворота**, то есть (x, y) , (w, h) , a .

Далее нам надо получить доступ к паре (w, h) . Поскольку в предыдущей программе результат функции `cv2.minAreaRect()` мы сохранили в переменную `rect`, то искомые значения будут находиться в ячейках массива по адресу `rect[1][0]` и `rect[1][1]`. Адресация элементов в массиве начинается с нуля. Перемножив эти значения, получим площадь текущего прямоугольника. Если значение площади отвечает заданному критерию, то этот контур нам интересен и мы его рисуем.

Таким образом, чтобы отфильтровывать только нужные контуры, необходимо модифицировать программу выделения прямоугольных контуров следующим фрагментом из двух строчек:

```
..
# вычисление площади
area = int(rect[1][0]*rect[1][1])
# если площадь больше указанного значения..
if area > 1500:
..
```

Вы, наверное, уже догадались, куда мы добавим эти две строчки. Естественно, сразу после строки с функцией `cv2.minAreaRect()`!

Прежде чем приступить к работе с программой, давайте сначала подготовим рисунок, на котором мы потренируемся фильтровать прямоугольные контуры. Для этого воспользуемся уже знакомым нам графическим редактором Paint, в котором создадим незамысловатый «натюрморт» из цветных прямоугольников случайных цветов и случайных размеров, примерно как на **рис. 2.32**. После чего сохраним файл с изображением в папку с программой, например под именем `rec_img1.jpg`.

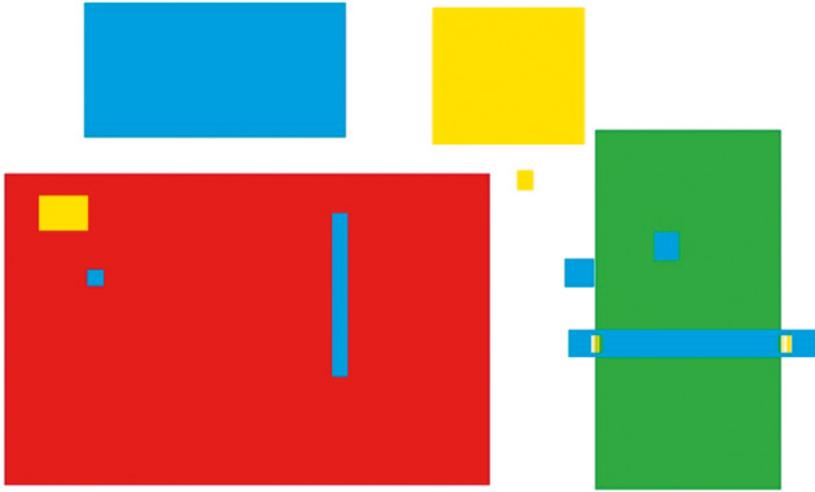


Рис. 2.32. Пример графического изображения для фильтрации прямоугольных контуров

Теперь займемся программой. Выведем на экран контуры больших прямоугольников светло-синего цвета. За основу берем предыдущую программу (см. Выделение прямоугольных контуров) и добавляем в нее две строки, приведенные выше. Затем подберем нужный цветовой фильтр. После несложного редактирования отступов некоторых строк вот что у нас получится:

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# задаем путь к файлу с картинкой,
# если файл с картинкой и файл с программой находятся
# в одной папке, то это будет просто имя графического
# файла
fileName = 'rec_img1.jpg'
# считываем данные графического файла в переменную
# image
image = cv2.imread(fileName)
# конвертируем исходное изображение в цветовую модель
# HSV,
# результат записываем в переменную hsv_img
hsv_img = cv2.cvtColor( image, cv2.COLOR_BGR2HSV )
```

```
# подбираем параметры цветового фильтра для выделения
# нашего объекта (указанные числовые значения могут
# отличаться)
hsv_min = np.array((90, 100, 100), np.uint8)
hsv_max = np.array((120, 255, 200), np.uint8)
# применяем цветовой фильтр к исходному изображению,
# результат записываем в переменную hsv_msk
hsv_msk = cv2.inRange( hsv_img, hsv_min, hsv_max )
# ищем контуры и записываем их в переменную contours
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST, для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours( hsv_msk,
                                     cv2.RETR_LIST,
                                     cv2.CHAIN_APPROX_SIMPLE)
# перебираем все найденные контуры в цикле
for icontour in contours:
#   ищем прямоугольник, результат записываем в rect
    rect = cv2.minAreaRect(icontour)
#   !!Вот здесь вставляются указанные выше две строки!!
#   вычисление площади
    area = int(rect[1][0]*rect[1][1])
#   если площадь больше указанного значения, эти
#   контуры выводим,
#   значение подбираем экспериментально
    if area > 1500:
#       поиск вершин прямоугольника, результат записываем
#       в box
        box = cv2.boxPoints(rect)
#       округление координат вершин, результат
#       записываем в box
        box = np.int0(box)
#       рисуем прямоугольник поверх исходного
#       изображения, цвет синий, толщина линии 3,
#       поскольку рисуем единственный объект [box],
#       остальные параметры опускаем
        cv2.drawContours( image, [box], -1,
                        (255,0,0),3)
# выводим итоговое изображение в окно contours
cv2.imshow('contours', image)
# ждем нажатия любой клавиши и закрываем все окна
```

```
cv2.waitKey()
```

```
cv2.destroyAllWindows()
```

Результат работы программы представлен на **рис. 2.33**. Как видите, фильтровать «графический сор» совсем несложно. Из представленного рисунка можно заметить, что не все светло-синие объекты выделены контурами. Меняя параметры цветового фильтра и значение площади прямоугольника в условии отбора, нетрудно выполнить эту же задачу для объектов других цветов. Таким же образом можно фильтровать прямоугольные контуры, учитывая соотношение сторон, периметр, задавая определенный диапазон значений по размерам и т. д.

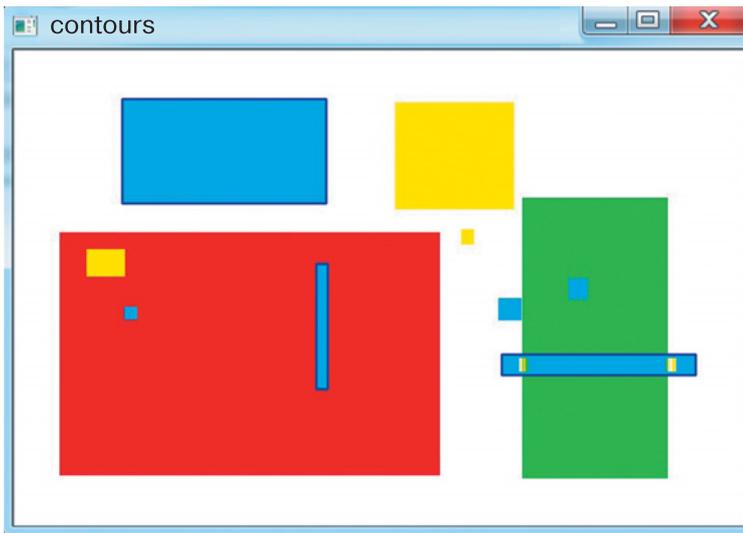


Рис. 2.33. Результат работы программы фильтрации прямоугольных контуров в тестовом изображении

ЗАДАНИЯ

1. Используя этот же рисунок, выполните фильтрацию контуров объектов желтого цвета и выведите в окне контур самого большого объекта.
2. Применяя наложение суммы масок, выделите из рисунка контуры объектов различных цветов, например красных и желтых объектов с отсевом мелких контуров и т. д.

Вычисление угла поворота прямоугольного контура

Как уже говорилось, эта задача имеет практическое применение. Определение угла поворота контура, описывающего прямоугольный объект, даст программе управления роботом-манипулятором данные, необходимые для успешного захвата объекта. Решение таких задач полезно, например, при программировании робота-сортировщика или робота-укладчика, когда требуется разложить в определенном порядке хаотично рассыпанные предметы или детали.

Итак, попробуем разобраться и понять, как решается данная задача. Если вы внимательно читали в одном из предыдущих разделов описание функции `cv2.minAreaRect()`, то, вероятно, обнаружили, что один ее параметр возвращает угол поворота прямоугольника. И теперь можете сказать, что тут все просто, нужно всего лишь взять значения этого параметра. Мы тоже так подумали. Нарисовали много прямоугольников, повернули их под разными углами, и вот что получили в результате (рис. 2.34):

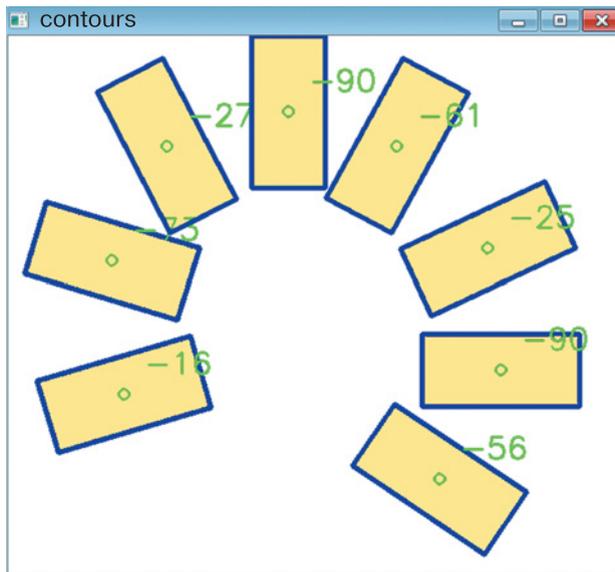


Рис. 2.34. Значения угла поворота прямоугольного контура, полученные из функции `cv2.minAreaRect()`

Складывается впечатление, что значения углов на рисунке определены хаотично. Отсутствует логическая зависимость определенного значения угла и изображения конкретного прямоугольника. Дело

в том, что функция `cv2.minAreaRect()` при определении угла наклона не вполне корректно различает соотношение сторон прямоугольника (длину и ширину в нашем понимании). Однако если на этот момент внимания не обращать и внимательно приглядеться к рис. 2.34, то окажется, что все значения вполне корректны. Но визуально нас такая картинка не устраивает. К тому же, если мы вдруг когда-нибудь соберемся программировать робота-манипулятора, нужно будет знать, как ориентировать захват: вдоль или поперек объекта. Поэтому мы пойдем другим путем!

Для начала определимся, относительно чего будем измерять угол наклона, а потом подумаем, как это можно сделать. У прямоугольника две пары сторон — одна длиннее другой; если они равны, то это уже квадрат. Чтобы было из чего исходить, примем за угол наклона угол между длинной стороной и направлением горизонтали. Естественно, для квадрата это может быть любая сторона. Далее нам придется обратиться к геометрии и вспомнить про векторы на плоскости.

Возьмем произвольный «повернутый» прямоугольник $ABCD$ на плоскости (рис. 2.35). Функция `cv2.boxPoints()` подскажет нам координаты четырех вершин этого прямоугольника, которые она возвращает в переменную `box`. Обозначим их как (x_1, y_1) , (x_2, y_2) , (x_3, y_3) , (x_4, y_4) для вершин A , B , C и D соответственно. Хотя функция `cv2.minAreaRect()` возвращает нам размеры прямоугольника (ширину и высоту), у нас все равно нет привязки этих размеров к его сторонам. Поэтому размеры сторон, например AB и AD , мы определим сами через координаты вершин прямоугольника. Для этого отрезки AB и AD представим как векторы. Координаты вектора определяются как разность координат его начала и конца. В результате получим

$$\overline{AB} = (x_2 - x_1, y_2 - y_1) = (x_b, y_b),$$

$$\overline{AD} = (x_4 - x_1, y_4 - y_1) = (x_d, y_d),$$

где x_b , y_b , x_d и y_d — условные обозначения координат соответствующих векторов.

Зная координаты векторов, можно определить их длину по теореме Пифагора. Но мы воспользуемся функцией `cv2.norm()`. Аргументом этой функции является массив `numpy`. Данная функция выполняет сложные математические расчеты; в общем виде она возвращает несколько видов матричных норм¹. В нашем случае такой

¹ Понятие нормы изучается в вузовском курсе математики, этому вопросу посвящены целые разделы. Мы воспользуемся евклидовой нормой, которая для нашего случая является арифметическим квадратным корнем из скалярного квадрата вектора, то есть его длиной, или модулем.

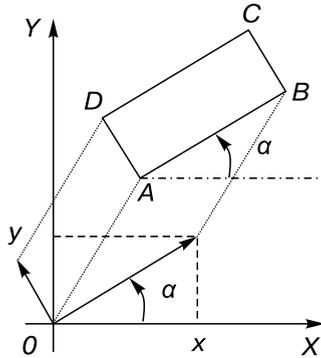


Рис. 2.35. Угол наклона прямоугольного контура

матрицей будет пара координат вектора, а возвращаемый результат — расстояние между точками, что в принципе нам и надо. Сравним длины векторов, далее работаем с вектором, имеющим наибольшую длину. Нам нужно найти угол между этим вектором и горизонталью (ось OX), на рис. 2.35 этот угол обозначен как α . Чтобы определить угол α , воспользуемся понятием косинуса угла из школьного курса геометрии.

ПОЯСНЕНИЕ

Рассмотрим прямоугольный треугольник ABC , в котором угол $C = 90^\circ$, сторона AB является гипотенузой, а стороны AC и BC — катетами. **Косинусом**¹ угла при вершине A называют отношение катета, прилежащего к этому углу, к гипотенузе: $\cos \alpha = AC/AB$. Если известен косинус угла, то значение угла определяется с помощью обратной функции — **арккосинуса** (\arccos).

Косинус угла α (см. рис. 2.35) равен отношению координаты x к длине вектора \overline{AB} . Для углов α , меньших 90° , косинус будет иметь положительное значение и изменяться от 1 до 0 при увеличении угла. Для тупых углов ($90^\circ + \alpha$) косинус будет иметь отрицательное значение и изменяться от 0 до -1 соответственно. Вычислив, таким образом, косинус угла α , воспользуемся функцией \arccos для определения значения самого угла α . Полученный угол и будет углом наклона нашего прямоугольного контура!

¹ См. также определение косинуса с помощью представления единичной окружности — URL: https://ru.wikipedia.org/wiki/Тригонометрические_функции (Дата обращения 20.12.2019).

Закончив с математикой, перейдем к рассмотрению алгоритма, на основе которого впоследствии напишем программу для определения угла наклона прямоугольного контура. Алгоритм состоит из такой последовательности действий:

- выполнить поиск прямоугольного контура;
- вычислить координаты векторов, являющихся сторонами прямоугольного контура;
- определить вектор наибольшей длины;
- вычислить угол между горизонталью и вектором наибольшей длины;
- вывести на экран результаты вычислений.

Подготовив предварительно тестовое изображение, например, как на рис. 2.34, сохраняем его в файл **rec_img2.jpg** и приступаем к написанию программы. За основу возьмем программный код из предыдущего параграфа (см. Фильтрация прямоугольных контуров). Для вычислений тригонометрических функций нам потребуется подключение встроенного пакета **math**, содержащего большое количество самых разнообразных математических функций. Вызов этого пакета добавит нам одну строку в начале программы. Результат выполнения программы показан на **рис. 2.36**.

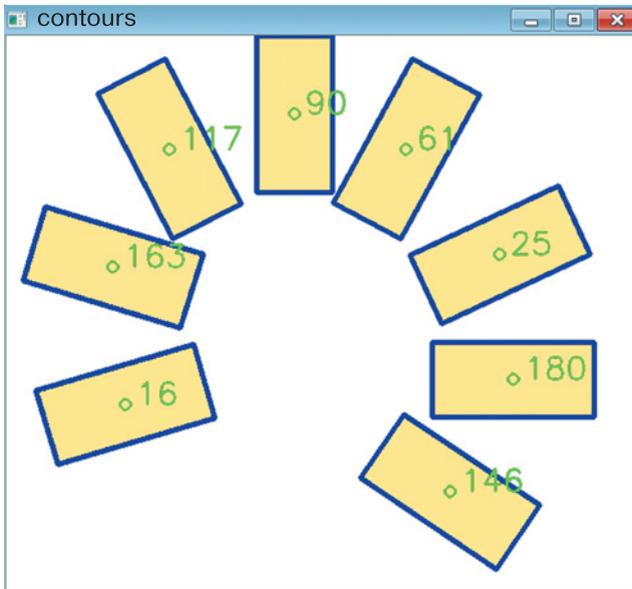


Рис. 2.36. Вычисленные значения угла поворота прямоугольных контуров

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# подключение библиотеки для математических расчетов
import math
# задаем путь к файлу с картинкой,
# если файл с картинкой и файл с программой находятся
# в одной папке, то это будет просто имя графического
# файла
fileName = 'rec_img2.jpg'
# считываем данные графического файла в переменную
# image
image = cv2.imread(fileName)
# конвертируем исходное изображение в цветовую модель
# HSV
# результат записываем в переменную hsv_img
hsv_img = cv2.cvtColor( image, cv2.COLOR_BGR2HSV )
# подбираем параметры цветового фильтра для выделения
# нашего объекта (указанные числовые значения могут
# отличаться)
hsv_min = np.array((0, 50, 50), np.uint8)
hsv_max = np.array((15, 255, 255), np.uint8)
# применяем цветовой фильтр к исходному изображению,
# результат записываем в переменную hsv_msk
hsv_msk = cv2.inRange( hsv_img, hsv_min, hsv_max )
# ищем контуры и записываем их в переменную contours
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST, для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours( hsv_msk,
                                       cv2.RETR_LIST,
                                       cv2.CHAIN_APPROX_SIMPLE)
# перебираем все найденные контуры в цикле
for icontour in contours:
# ищем прямоугольник, результат записываем в rect
rect = cv2.minAreaRect(icontour)
# вычисление площади прямоугольного контура
area = int(rect[1][0]*rect[1][1])
# отсекаем ложные контуры, если они вдруг появятся
if area > 2500:
```

```

# поиск вершин прямоугольника, результат записываем
# в box
    box = cv2.boxPoints(rect)
# округление координат вершин, результат
# записываем в box
    box = np.int0(box)
# рассчитываем координаты первого вектора
    vec1 = np.int0((box[1][0] - box[0][0],
                    box[1][1] - box[0][1]))
# рассчитываем координаты второго вектора
    vec2 = np.int0((box[2][0] - box[1][0],
                    box[2][1] - box[1][1]))
# мы не знаем, какой вектор больше, поэтому
# думаем, что это будет первый вектор,
# сохраняем его в used_vec
    used_vec = vec1
# если длина второго вектора больше первого,
    if cv2.norm(vec2) > cv2.norm(vec1):
# значит, в used_vec сохраним длину второго
# вектора
        used_vec = vec2
# записываем координаты центра прямоугольника
    center = (int(rect[0][0]),int(rect[0][1]))
# рассчитываем угол наклона прямоугольника

    angle = 180.0/math.pi*math.acos(used_vec[0]/
                                    cv2.norm(used_vec))
# рисуем прямоугольник
    cv2.drawContours( image, [box], -1,
                    (255,0,0),3)
# рисуем метку-окружность в центре
# прямоугольника
    cv2.circle(image, center, 5, (0,255,0), 2)
# выводим рядом с прямоугольником значение угла
# наклона
    cv2.putText(image, "%d" % int(angle),
                (center[0]+10, center[1]),
                cv2.FONT_HERSHEY_SIMPLEX,1, (0,255,0),2)
# выводим итоговое изображение в окно contours
cv2.imshow('contours', image)
# ждем нажатия любой клавиши и закрываем все окна

```

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

После запуска программы мы получили совсем другой результат, нежели был ранее. Теперь возле каждого прямоугольника подписано вполне понятное значение угла поворота относительно горизонтали (см. рис. 2.36).

ЗАДАНИЯ

1. Подготовьте в графическом редакторе рисунок с прямоугольниками различных цветов и размеров, а также углами поворота. Полученный рисунок сохраните в формате **.jpg** в папке с программой.
2. Произведите выделение прямоугольного контура и вычисление угла его поворота на подготовленном изображении, меняя числовые параметры для переменных **hsv_min**, **hsv_max**, для каждого цвета.
3. Применяя наложение суммы масок, выделите из рисунка контуры объектов различных цветов, например красных и желтых объектов и т. д.

Выделение и фильтрация эллиптических контуров

Для выделения эллиптических контуров в OpenCV применяется функция **cv2.fitEllipse()**. Как и функция **cv2.inAreaRect()**, функция **cv2.fitEllipse()** не сможет отличить на картинке объект с действительно эллиптическим контуром от объекта с прямоугольным контуром. Эта функция лишь пытается *вписать* эллипс в любой контур с количеством точек, большим или равным 5. Чуть позже мы увидим, как функция **cv2.fitEllipse()** попытается *описать* объект неэллиптической формы. Результатом данной функции, согласно документации¹, является повернутый прямоугольник, в который вписан эллипс.

¹ OpenCV 2.4.13.7 documentation. Structural Analysis and Shape Descriptors. // URL: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html (Дата обращения 18.12.2019).

В общем виде функция `cv2.fitEllipse()` записывается следующим образом:

```
cv2.fitEllipse(contour)
```

где **contour** — это контур, представляющий собой двухмерный массив **numpy**, в который будет вписан эллипс.

Чтобы нарисовать найденный эллиптический контур, воспользуемся функцией `cv2.ellipse()`. Общий вид данной функции имеет два определения¹. Нам понадобится определение с альтернативным представлением эллипса, когда отображаемый эллипс вписан в повернутый прямоугольник:

```
cv2.ellipse(img, box, color[, thickness[, lineType]])
```

где **img** — переменная с изображением;

box — альтернативное представление эллипса (эллипс вписан в повернутый прямоугольник);

color — цвет эллипса, задается комбинацией значений (B, G, R);

thickness — толщина линии в пикселях (необязательный параметр);

lineType — тип линии (необязательный параметр).

Задача фильтрации реализуется путем вычисления длины эллиптического контура. Далее, указав необходимые условия, мы можем отображать нужные нам контуры. Для вычисления длины контура применим функцию `len()`. Эта функция возвращает число элементов в массиве. Если в качестве аргумента функции `len()` мы укажем конкретный контур (а он представляет собой массив точек!), то получим его длину в точках.

Для проверки работы программы сначала подготовим тестовую картинку, на которой будут изображены различные эллиптические объекты и какой-нибудь многоугольник. Мы будем работать с картинкой, изображенной на **рис. 2.37**. Сохраним ее в файл под именем **ellips.jpg**

Алгоритм программы поиска эллиптических контуров очень похож на алгоритм поиска прямоугольных контуров, приведем его еще раз в словесной форме:

1. Получить доступ к данным графического файла.
2. Перевести изображение в цветное HSV-пространство.

¹ OpenCV 2.4.13.7 documentation. Drawing functions. // URL: https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html (Дата обращения 18.12.2019).



Рис. 2.37. Пример графического изображения для выделения эллиптических контуров

3. Выделить объект с помощью цветового HSV-фильтра.
4. Выполнить поиск всех контуров.
5. Начать цикл по всем найденным контурам.
6. Найти эллипс, описывающий текущий контур.
7. Отобразить полученный эллипс на исходном изображении.
8. Закончить цикл.
9. Вывести на экран окно с исходным изображением.

Ниже представлена программа выделения эллиптических контуров:

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# задаем путь к файлу с картинкой,
# если файл с картинкой и файл с программой находятся
# в одной папке, то это будет просто имя графического
# файла
fileName = 'ellips.jpg'
# считываем данные графического файла в переменную
image
image = cv2.imread(fileName)
# конвертируем исходное изображение в цветовую модель
```

```

# HSV
# результат записываем в переменную hsv_img
hsv_img = cv2.cvtColor( image, cv2.COLOR_BGR2HSV )
# подбираем параметры цветового фильтра для выделения
# нашего объекта (указанные числовые значения могут
# отличаться)
hsv_min = np.array((60, 10, 10), np.uint8)
hsv_max = np.array((180, 255, 255), np.uint8)
# применяем цветовой фильтр к исходному изображению,
# результат записываем в переменную hsv_msk
hsv_msk = cv2.inRange( hsv_img, hsv_min, hsv_max )
# ищем контуры и записываем их в переменную contours
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST, для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours( hsv_msk,
                                       cv2.RETR_LIST,
                                       cv2.CHAIN_APPROX_SIMPLE)
# перебираем все найденные контуры в цикле
for icontour in contours:
#   выбираем контуры с длиной больше 40 точек
    if len(icontour)>40:
#       записываем в переменную ellipse
#       отвечающий условию контур в форме эллипса
        ellipse = cv2.fitEllipse(icontour)
#       отображаем найденный эллипс
        cv2.ellipse(image, ellipse, (255,0,255), 2)
# выводим итоговое изображение в окно contours
cv2.imshow('contours', image)
# выводим результат фильтрации изображения в окно HSV
# cv2.imshow('hsv', hsv_msk)
# ждем нажатия любой клавиши и закрываем все окна
cv2.waitKey()
cv2.destroyAllWindows()

```

Результат работы программы представлен на **рис. 2.38**. Как видим, на рисунке эллиптическими контурами описаны не только эллиптические объекты. И в то же время не все эллиптические объекты помечены контуром. Интересно, почему?

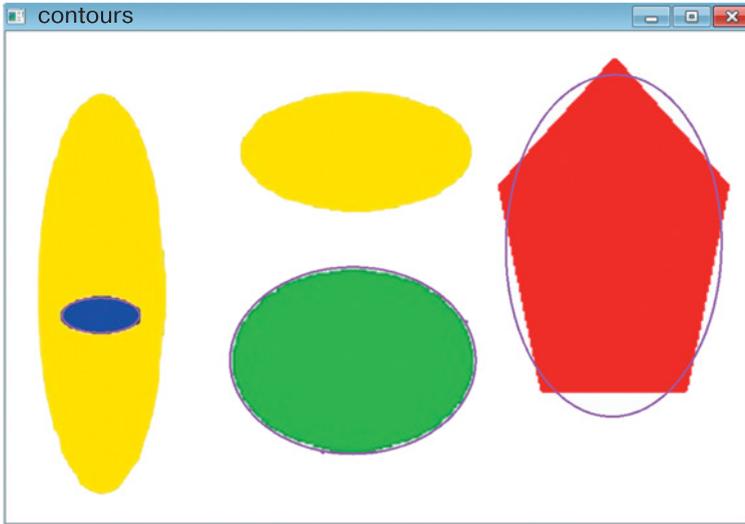


Рис. 2.38. Выделение эллиптических контуров в тестовом изображении

Если приглядимся повнимательнее, то увидим, что желтые овалы остались без контуров. Заглянув в параметры цветового фильтра, мы обнаружим, что желтый цвет находится за пределами границ фильтра. В этом можно убедиться, если раскомментировать в программе четвертую строчку снизу и запустить весь код на исполнение. Теперь мы увидим окно, в котором нет «следов» желтых овалов (**рис. 2.39**).



Рис. 2.39. Результаты фильтрации цветных объектов в тестовом изображении: следов желтых объектов нет

ЗАДАНИЯ

1. Подготовьте в графическом редакторе рисунок с несколькими объектами (5–6 единиц) разных цветов, размеров и форм. Сохраните полученный рисунок в формате **.jpg** в папке с программой.
2. Выполните поиск и отображение эллиптического контура объекта, подбирая значение в функции **len()**: а) самого малого размера; б) самого большого размера.

Выделение контуров дорожных знаков в видеопотоке

Настало время применить на практике первые полученные знания из контурного анализа. Поставим себе задачу: обозначить контурами некоторые дорожные знаки на фрагменте записи с настоящего автомобильного видеорегистратора!

Для этого воспользуемся программным кодом, работа которого была рассмотрена в разделе «Суммирование цветовых масок». Это позволит существенно упростить решение поставленной задачи, хотя на фоне того, что мы уже изучили, алгоритм задачи не такой уж и сложный.

Вот последовательность его действий в кратком виде:

1. Подключить необходимые библиотеки и получить доступ к видеофайлу.
2. Найти дорожные знаки с помощью цветовых HSV-фильтров.
3. Обозначить найденные объекты прямоугольными контурами с помощью функции **minAreaRect()**.
4. Выполнить фильтрацию найденных контуров по площади, отсеив «графический шум», и с помощью функции **cv2.drawContours()** нарисовать контуры искоемых объектов.
5. Вывести результат работы программы.

Как видите, каждый из перечисленных выше пунктов по отдельности мы уже делали, а теперь посмотрим, как этот алгоритм будет реализован в программном коде. Чтобы регулировать скорость просмотра кадров, дополнительно подключим библиотеку **time**, в которой есть функция ожидания **time.sleep()**. Аргументом этой функ-

ции является числовой параметр в секундах, устанавливающий время ожидания.

При подборе параметров граничных значений цветового фильтра HSV опираемся на рис. 2.18, а также выкладки, приведенные на с. 55–56.

Ниже представлен программный код для выделения дорожных знаков на видео с помощью прямоугольных контуров:

```
# ПОДКЛЮЧАЕМ БИБЛИОТЕКИ И ПОЛУЧАЕМ ДОСТУП
# К ВИДЕОФАЙЛУ
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# подключение библиотеки функций работы со временем,
# она будет нужна для изменения скорости смены кадров
import time
# связываем видеопоток файла video.avi с переменной
# capImg
capImg = cv2.VideoCapture('video.avi')

# ИЩЕМ ЗНАКИ С ПОМОЩЬЮ HSV-ФИЛЬТРОВ
# открываем файл с видео
while (capImg.isOpened()):
    # получаем кадр из видеопотока файла,
    # кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
    # если кадры закончились, то прерываем цикл
    if frame is None:
        break
    # переводим кадр в цветовое пространство HSV
    frame_hsv = cv2.cvtColor(frame,
                               cv2.COLOR_BGR2HSV)
    # делаем вырезки области кадра, где ожидаются знаки
    crop_frame = frame[160:270, 150:650]
    crop_frame_hsv = frame_hsv[160:270, 150:650]
    # задаем нижние и верхние границы цветовых фильтров
    # с помощью массивов numpy
    # цвет 0...180
    # насыщенность 0 - блеклый, 255 - насыщенный
    # яркость 0 - темный, 255 - светлый
```

```
# фильтр для синего цвета
low_Blue = np.array([105, 150, 0],
                    dtype = "uint8")
high_Blue = np.array([135, 255, 255],
                    dtype = "uint8")

# фильтр для желтого цвета
low_Yel = np.array([25, 95, 100],
                  dtype = "uint8")
high_Yel = np.array([35, 255, 255],
                  dtype = "uint8")

# фильтр для красного цвета
# красный цвет содержит две области
# красный в сторону оранжевой области
low_Red_O = np.array([0, 85, 110],
                    dtype = "uint8")
high_Red_O = np.array([5, 165, 155],
                    dtype = "uint8")

# красный в сторону фиолетовой области
low_Red_V = np.array([165, 55, 40],
                    dtype = "uint8")
high_Red_V = np.array([180, 105, 120],
                    dtype = "uint8")

# применяем маску по каждому цвету к фрагменту
# кадра для выделения синего цвета
blue_mask = cv2.inRange(crop_frame_hsv, low_Blue,
                       high_Blue)

# для выделения желтого цвета
yel_mask = cv2.inRange(crop_frame_hsv, low_Yel,
                      high_Yel)

# для выделения красного накладываются две маски
red1_mask = cv2.inRange(crop_frame_hsv,
                       low_Red_O, high_Red_O)
red2_mask = cv2.inRange(crop_frame_hsv,
                       low_Red_V, high_Red_V)

# вычисляем полную маску
# полная маска представляет собой сумму всех масок
full_mask = red1_mask + red2_mask + blue_mask +
            yel_mask

# ЗАКЛЮЧАЕМ В ПРЯМОУГОЛЬНЫЕ КОНТУРЫ НАЙДЕННЫЕ ОБЪЕКТЫ
# ищем контуры и записываем их в переменную
# contours
```

```
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST, для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours(full_mask.
copy(),
cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
# перебираем все найденные контуры в цикле
for icontour in contours:
# ищем прямоугольник, результат записываем
# в rect
rect = cv2.minAreaRect(icontour)
# вычисление площади прямоугольного контура
area = int(rect[1][0]*rect[1][1])

# ФИЛЬТРУЕМ МЕЛКИЕ КОНТУРЫ
# отсекаем ложные контуры, если они вдруг
# появятся
if area > 100:
# поиск вершин прямоугольника, результат
# записываем в box
box = cv2.boxPoints(rect)
# округление координат вершин, результат
# записываем в box
box = np.int0(box)
# рисуем прямоугольник
cv2.drawContours(crop_frame, [box], 0,
(0, 255, 255), 2)

# ВЫВОДИМ РЕЗУЛЬТАТЫ НА ЭКРАН
# выводим изображения в окнах на экран
cv2.imshow("video_mask", full_mask)
cv2.imshow("video_frame", frame)
cv2.imshow("Crop_frame", crop_frame)
# задержка по времени, аргумент подбираем
# экспериментально
time.sleep(0.1)
# организуем выход из цикла по нажатию клавиши,
# ждем 30 миллисекунд нажатия, записываем код
# нажатой клавиши
key_press = cv2.waitKey(30)
```

```
# если код нажатой клавиши совпадает с кодом «q»,
  if key_press == ord('q'):
      break
# освобождаем память от переменной cap
cap.release()
# закрываем все окна opencv
cv2.destroyAllWindows()
```

На **рис. 2.40** представлен результат работы программы, где на одном из кадров запечатлен момент с обнаружением дорожного знака «Пешеходный переход». Сразу видно, что даже простейшая фильтрация контуров по размерам позволяет игнорировать мелкие артефакты, которые появляются при поиске цветных объектов. Чтобы повысить качество поиска дорожных знаков или других объектов на видео, надо понимать, что оно напрямую зависит от качества самого видео. Поэтому вполне допустимо, что на видео не очень хорошего качества наша программа, кроме дорожных знаков, найдет что-то еще.

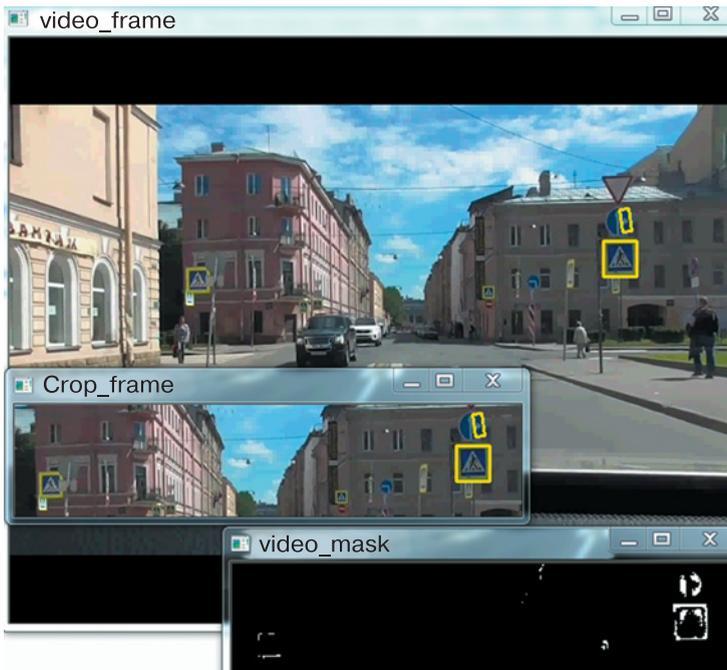


Рис. 2.40. Выделение в потоке видео дорожного знака прямоугольным контуром (Источник: кадр из ролика https://www.youtube.com/watch?time_continue=2&v=eC6KtIYOIf4&feature=emb_logo)

Итоги главы 2

В главе 2 мы познакомились с математическим пакетом `numpy` и библиотекой компьютерного зрения `OpenCV`. Изучили методы выполнения основных операций с изображением в `OpenCV`. Сделали первые шаги в поиске объектов по цвету, применяя цветовые фильтры. Узнали об особенностях представления цвета в пространствах цветов `RGB` и `HSV`. С помощью методов работы с видеопотоком научились настраивать цветовые фильтры для поиска цветных объектов в кадре. Познакомились с простейшими задачами контурного анализа и научились выделять найденные объекты прямоугольными и эллиптическими контурами.

Теперь мы знаем:

1. Адрес официального сайта разработчика библиотеки `OpenCV`.
2. Порядок установки `OpenCV-Python` на персональный компьютер.
3. Методы открытия, трансформации, фрагментации и сохранения данных графических файлов.
4. Способ поиска цветного объекта на изображении путем наложения цветового фильтра.
5. Понятие цветового пространства.
6. Способы представления цвета в пространствах `RGB` и `HSV`.
7. Понятие момента черно-белого (`bitmap`) изображения.
8. Методы определения координат найденного цветного объекта и вывод их значений в графическое окно.
9. Методы чтения и отображения данных видеопотока из `web-камеры` и `видеофайла`.
10. Методы поиска единичного цветного объекта или группы цветных объектов в потоке видеоданных.
11. Методы создания простейшего графического интерфейса `OpenCV` для настройки цветового фильтра.
12. Методы поиска и выделения контуров найденных цветных объектов.
13. Методы отображения и фильтрации прямоугольных и эллиптических контуров.
14. Методы вычисления угла поворота прямоугольного контура.

Теперь мы умеем:

1. Скачать и установить библиотеку `OpenCV-Python`.
2. Читать, отображать и сохранять данные графических файлов средствами `OpenCV-Python`.
3. Масштабировать, фрагментировать, поворачивать и зеркально отражать графическое изображение средствами `OpenCV-Python`.

4. Находить цветной объект в картинке путем построения цветового фильтра в цветовом RGB- или HSV-пространстве.
5. Находить и выводить в графическое окно координаты найденного цветного объекта.
6. Захватывать и отображать кадры видеопотока из web-камеры или видеофайла.
7. Находить и отображать цветной объект в потоке видеок кадров.
8. Суммировать цветовые фильтры для поиска и отображения группы цветных объектов в потоке видеок кадров.
9. Создавать интерфейсное окно OpenCV для настройки цветового фильтра.
10. Находить и отображать контуры искомым цветных объектов.
11. Находить и фильтровать прямоугольные и эллиптические контуры по размеру.
12. Рассчитывать угол поворота прямоугольного контура.

Глоссарий к главе 2

Здесь собраны основные термины и понятия, а также функции языка Python, встречающиеся в главе 2.

ASCII (American Standard Code for Information Interchange) — название кодировочной таблицы, в которой распространенным печатным и непечатным символам сопоставлены числовые коды.

AVI (Audio Video Interleave) — медиаконтейнер, разработанный компанией Microsoft для пакета Video for Windows. Файл AVI может содержать видео- и аудиоданные, сжатые с помощью разных комбинаций видео- и аудиокодеков, что позволяет синхронно воспроизводить видео со звуком.

BGR — цветовое пространство: синий (**B**lue), зеленый (**G**reen), красный (**R**ed); каждая компонента может принимать значение в диапазоне 0–255.

break — команда, прерывает выполнение любого цикла.

cv2.boxPoints() — функция, возвращает массив из вещественных координат четырех точек — вершин прямоугольника.

cv2.createTrackbar() — функция, создает графический компонент «ползунок с указателем».

cv2.cvtColor() — функция, конвертирует исходное изображение из одного цветового пространства в другое и возвращает сконвертированное изображение.

cv2.destroyAllWindows() — функция, закрывает все графические окна.

cv2.drawContours() — функция, отображает контуры, найденные функцией **cv2.findContours()**.

- cv2.ellipse()** — функция, отображает найденный эллиптический контур.
- cv2.findContours()** — функция, возвращает замкнутые контуры графических областей в виде массива точек.
- cv2.fitEllipse()** — функция, возвращает повернутый прямоугольник, в который вписан эллипс длиной не менее 5 точек.
- cv2.flip()** — функция, возвращает зеркальную копию исходного изображения.
- cv2.getRotationMatrix2D()** — функция, возвращает двумерный объект, в котором будет размещено графическое изображение.
- cv2.getTrackbarPos()** — функция, считывает положение указателя ползунка.
- cv2.imread()** — функция, считывает массив данных из файла с изображением.
- cv2.imshow()** — функция, отображает графические данные на экран.
- cv2.imwrite()** — функция, сохраняет изображение в графический файл.
- cv2.inRange()** — функция, возвращает черно-белое изображение, где белыми пикселями обозначены области, цвета которых попали в заданный цветовой фильтр.
- cv2.minAreaRect()** — функция, ищет прямоугольный контур минимальной площади, способный описать заданный замкнутый контур.
- cv2.moments()** — функция, вычисляет моменты графической области белого цвета на черно-белом изображении.
- cv2.namedWindow()** — функция, создает графическое окно.
- cv2.putText()** — функция, выводит в графическое окно OpenCV текстовое сообщение.
- cv2.resize()** — функция, возвращает трансформированное изображение.
- cv2.VideoCapture()** — функция, обеспечивает идентификацию и доступ к потоку видеоданных.
- cv2.waitKey()** — функция, ожидает нажатия любой клавиши с клавиатуры при пустом аргументе или нажатия клавиши, соответствующей символу в одинарных кавычках, например 'п'.
- cv2.warpAffine()** — функция, выполняет поворот изображения и возвращает новое преобразованное изображение (в случае трансформации и поворота).
- HSV(HSB)** — цветовое пространство: цветовой тон (**Hue**), насыщенность (**Saturation**) и значение или яркость (**Value** или **Brightness**). Под цветовым тоном понимается именно цвет (0–180). Насыщенность характеризует близость цвета к белому (0–255). Значение или яркость представляет как общую яркость точки или цвета (0–255).
- image.shape[i]** — метод, команда, возвращает ширину (при $i = 1$) или высоту (при $i = 0$) исходного изображения **image**.

ord() — функция, возвращает код указанного символа.

pass — оператор-заглушка, не выполняет никаких действий.

Region Of Interest (ROI) — определенная пользователем прямоугольная область на рисунке.

Арккосинус — тригонометрическая функция, обратная функции косинуса и возвращающая значение угла, если известен косинус.

Библиотека OpenCV (Open Source Computer Vision Library) — библиотека компьютерного зрения с открытым исходным кодом.

Косинус острого угла α в прямоугольном треугольнике — это отношение прилежащего катета AC к гипотенузе AB : $\cos \alpha = AC/AB$.

Нейронная сеть — в нашем случае это искусственная нейронная сеть, представляет собой математическую модель, которая может быть реализована в программном коде или в аппаратном устройстве. Нейронные сети используются для решения задач адаптивного управления, распознавания образов, в системах искусственного интеллекта и т. д.

Список литературы к главе 2

1. Open Source Computer Vision Library // Официальный сайт разработчика OpenCV. URL: <https://opencv.org/>
2. Releases OpenCV // Официальный сайт разработчика OpenCV. URL: <https://opencv.org/releases.html>
3. Install OpenCV-Python in Windows // Официальный сайт разработчика OpenCV. URL: https://docs.opencv.org/3.3.1/d5/de5/tutorial_py_setup_in_windows.html
4. Установка OpenCV в Windows 10 с помощью Python 3.6 и Anaconda 3.6 // Ru Puthon. URL: <https://www.rupython.com/opencv-windows-10-python-3-6-anaconda-3-6-10672.html>
5. Установка OpenCV 3.1.0 для Python 3.5 в Windows. URL: <http://san-tit.blogspot.com/2016/03/opencv-310-python-35-windows.html>
6. Центр загрузки. Распространяемый пакет Visual C++ для Visual Studio 2015 // Официальный сайт Microsoft. URL: <https://www.microsoft.com/RU-RU/download/details.aspx?id=48145>
7. Python Software Foundation // Официальный сайт разработчика Python. URL: <https://www.python.org/ftp/python/2.7.10/python-2.7.10.msi>
8. Project Activity OpenCV // SourceForge is an Open Source community resource. URL: <https://sourceforge.net/projects/opencvlibrary/>

Программирование Raspberry Pi

Микрокомпьютеры Raspberry Pi

Raspberry Pi¹ — одноплатный компьютер компактного размера [1]. С целью обеспечения взаимодействия с периферийными устройствами его плата имеет разъем HDMI для подключения монитора, разъем для подключения DSI-экрана, разъем для подключения CSI-камеры, USB-порты для подключения USB-устройств, GPIO-разъем для подключения низкоуровневой периферии, а также Ethernet-порт для подключения к сети. Кроме того, последние поколения Raspberry Pi снабжены интегрированными модулями беспроводной связи Wi-Fi и Bluetooth.

ПОЯСНЕНИЕ

HDMI (High Definition Multimedia Interface) — интерфейс передачи мультимедийных данных высокой четкости. Позволяет передавать цифровой видеосигнал высокого разрешения и многоканальное цифровое аудио с защитой от копирования. Пропускная способность канала HDMI от 4,9 до 48 Гбит/с.

DSI (Display Serial Interface) — специализированный интерфейс для работы с жидкокристаллическими (ЖК) экранами. Максимальная пропускная способность интерфейса DSI составляет 1 Гбит/с.

CSI (Camera Serial Interface) — компактный специализированный интерфейс для работы с цифровой камерой. CSI-интерфейс обладает пропускной способностью до 2,5 Гбит/с и идеально под-

¹ Raspberry Pi // Официальный сайт разработчика Raspberry Pi. URL: <https://www.raspberrypi.org> (Дата обращения 25.12.2018).

ходит для встраиваемых систем машинного зрения, где важным фактором является низкое ресурсопотребление.

GPIO (General-Purpose Input/Output) — интерфейс для связи между компонентами компьютерной системы и различными периферийными устройствами. Разъем GPIO содержит 40 контактов. Контакты GPIO можно задействовать как в качестве входных, так и выходных (режим работы контактов конфигурируется на программном уровне).

Ethernet — технология пакетной передачи данных в компьютерных сетях. Стандарт Ethernet определяет проводные соединения компьютерных сетей, физически реализованные с помощью коаксиального кабеля, витой пары или оптического кабеля. На платах Raspberry Pi установлен стандартный разъем RJ45, применяемый для подключения витой пары.

Wi-Fi — технология беспроводной локальной сети для передачи цифровых потоков данных по радиоканалам с частотой 2,4–2,5 ГГц. Радиус действия до 100 м, пропускная способность до 6,77 Гбит/с.

Bluetooth — открытый стандарт беспроводной связи с низким энергопотреблением для передачи данных между совместимыми устройствами. Радиус действия до 100 м, пропускная способность до 24 Мбит/с.

Первые модели плат из партии в 10 тыс. экземпляров были выпущены в феврале 2012 г. С того времени было произведено несколько версий Raspberry. В феврале 2016 г. появилась самая мощная модель — плата Raspberry Pi 3, которая имеет встроенные Wi-Fi- и Bluetooth-модули, оснащена 4-ядерным 64-битным процессором



Рис. 3.1. Одноплатный компьютер Raspberry Pi 3 model B

ARM Cortex-A53 с тактовой частотой 1,2 ГГц и оперативной памятью объемом в 1 Гб (**рис. 3.1**). Более подробные обзор и описание Raspberry Pi можно посмотреть здесь^{1, 2} [2, 3].

Благодаря высоким техническим характеристикам и небольшой цене Raspberry Pi 3 отлично подходит для разработки систем, оснащенных компьютерным зрением. Технические характеристики Raspberry Pi 3 представлены в таблице:

Микрокомпьютер Raspberry Pi 3	Broadcom BCM2837
Процессор	ARM Cortex-A53 (4 ядра)
Графический процессор	Broadcom VideoCore IV
Оперативная память	1 Гб LDDR2
Встроенные адаптеры	10/100 Мбит Ethernet Bluetooth 4.1 LE Wi-Fi 802.11n
Порты	4 × USB 2.0 HDMI Аудиовыход, 3,5 мм 40-pin GPIO Camera Serial Interface (CSI) Display Serial Interface (DSI)
Разъем	microSD

Установка операционной системы Raspbian

Прежде чем начать работу с Raspberry Pi, необходимо подготовить карту памяти microSD с операционной системой. Этот одноплатный компьютер не имеет встроенной постоянной памяти, поэтому хранение всех файлов и развертывание операционной системы осуществляются на карте памяти. Для Raspberry Pi специально разработана операционная система Raspbian³ [4]. Ее установка не пред-

¹ Raspberry Pi 3. Обзор и начало работы // Портал о микроконтроллере Raspberry Pi. URL: <https://myraspberrypi.ru/obzor-raspberry-pi-3-odnoplattnik,-kotoryij-stal-polnoczennyim-kompyuterom.html> (Дата обращения 25.12.2018).

² Raspberry Pi 3. Обзор и начало работы. URL: <https://dmitrysnotes.ru/raspberry-pi-3-obzor-i-nachalo-raboty> (Дата обращения 25.12.2018).

³ Raspberry Pi. Downloads // Официальный сайт разработчика Raspberry Pi. URL: <https://www.raspberrypi.org/downloads/> (Дата обращения 25.12.2018).

ставляет особых трудностей, а после установки она сразу готова к работе. Надо отметить, что, кроме Raspbian, существует множество других операционных систем [4], работающих на Raspberry Pi.

Для установки операционной системы Raspbian в комплекте с библиотекой OpenCV рекомендуется применять карту памяти емкостью не менее 16 Гб. В этом случае можно быть уверенным, что хватит места и для собственных проектов. Однако не все карты памяти объемом 16 Гб подойдут для работы с Raspberry Pi. Выбрать подходящую карту вам помогут результаты тестирования¹ [5].

ПОЯСНЕНИЕ

microSD — один из форм-факторов энергонезависимых ультракомпактных карт памяти (флеш-память) размером $11 \times 15 \times 1$ мм, весом около 1 г, относящихся к семейству карт памяти формата SD (Secure Digital). Официально стандарт microSD был утвержден в 2005 г. ассоциацией SD Card Association.

При наличии подходящей карты microSD можно приступить к установке операционной системы Raspbian. Для этих целей понадобится компьютер и CardReader (обычно в большинстве ноутбуков он встроенный) для работы с картой памяти. Существует несколько способов установки операционной системы для Raspberry Pi. Эти способы доступно и подробно описаны в Интернете [6–8].

Воспользуемся одним из способов — с помощью установщика **NOOBS (New Out Of Box Software)**. Сначала нужно скачать архив с NOOBS на диск компьютера. Для этого перейдем на официальный сайт разработчиков Raspberry Pi и посетим страницу загрузки NOOBS² [9]. Загрузка архива начнется после нажатия кнопки, указанной стрелкой на **рис. 3.2**.

После загрузки архива извлекаем его содержимое в отдельную папку, а затем содержимое этой папки скопируем в корневую папку карты microSD. Карта должна быть предварительно отформатирована с использованием файловой таблицы FAT32 и стандартным размером

¹ RPi SD cards // URL: https://elinux.org/RPi_SD_cards (Дата обращения 25.12.2018).

² Raspberry Pi. Downloads. NOOBS // Официальный сайт разработчика Raspberry Pi. URL: <https://www.raspberrypi.org/downloads/noobs/>

После запуска установщика NOOBS на экран будет выведено окно, вид которого зависит от версии NOOBS, с выбором установки операционной системы. Окно должно выглядеть примерно как на **рис. 3.4**.

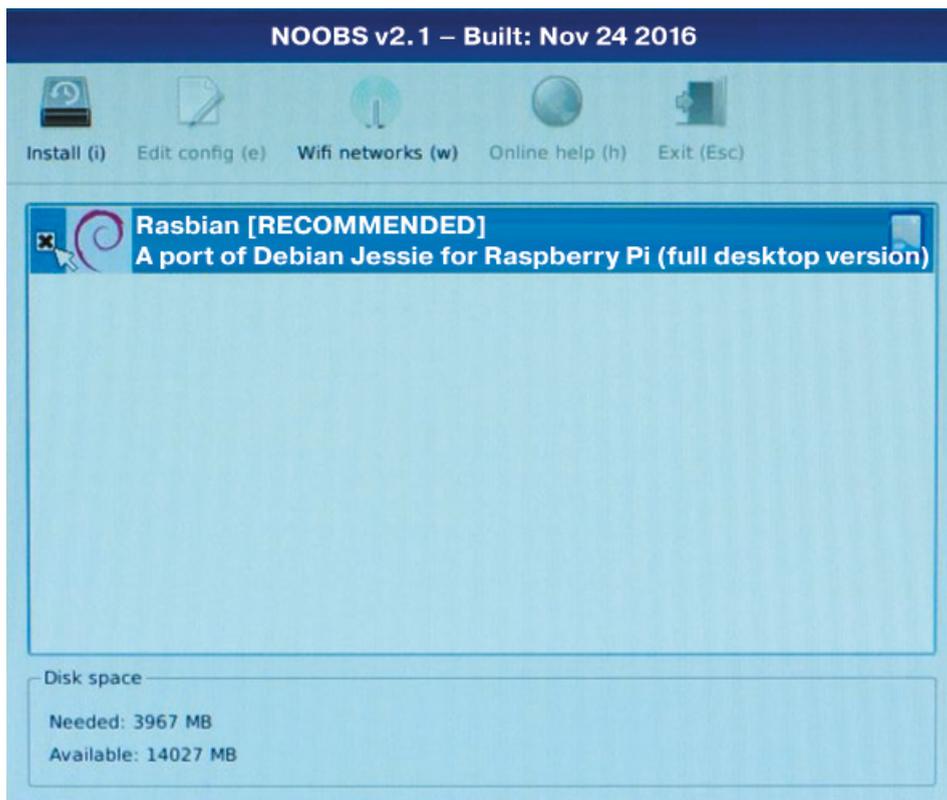


Рис. 3.4. Окно NOOBS выбора установки операционной системы

Выбрав соответствующую операционную систему, нажмите кнопку Install. После нажатия кнопки NOOBS запускает процесс установки выбранной операционной системы — у нас это Raspbian. Процесс установки Raspbian занимает 10–15 мин.

По окончании процедуры установки операционной системы Raspberry Pi полностью готов к работе (**рис. 3.5**). К этому моменту в системе уже установлены различные полезные программы:

- среда программирования на языке Python версий 2 и 3;
- визуальная среда программирования на языке Scratch;
- средства разработки Java-приложений;

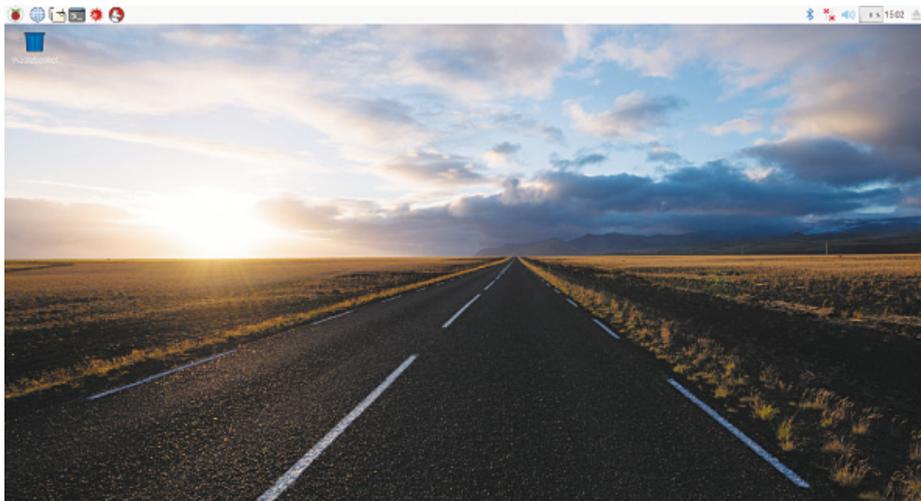


Рис. 3.5. Рабочий стол Raspbian

- пакет Mathematica;
- пакет офисных приложений LibreOffice;
- VNC Viewer;
- браузер Chromium и т. д.

Установка OpenCV на Raspberry Pi 3

Следующим действием является установка библиотеки OpenCV на Raspberry Pi. Здесь процедура установки OpenCV несколько отличается от аналогичной процедуры, описанной в главе 2.

Установка библиотеки OpenCV потребует внушительного объема свободной памяти на карте microSD — около 7 Гб. Поэтому неудивительно, что для установки OpenCV была рекомендована карта памяти емкостью не менее 16 Гб.

Шаг 1. Перед началом установки OpenCV следует обновить операционную систему. При этом Raspberry Pi должен иметь подключение к сети Интернет. Для обновления системы в строке терминала по очереди вводятся на исполнение команды¹:

¹ В терминале команда пишется одной строкой. Однако формат книги не позволяет выполнить это требование, поэтому некоторые команды занимают несколько строк. Во избежание путаницы в этом случае между командами делается интервал.

```
sudo apt-get update
sudo apt-get dist-upgrade
```

Далее требуется перезагрузить систему.

ПОЯСНЕНИЕ

Терминал — текстовое окно для ввода команд в операционной системе Raspbian; является аналогом командной строки операционной системы Windows. Вызов терминала осуществляется сочетанием клавиш Ctrl+Alt+T либо через меню приложений.

Шаг 2. Для работы с библиотекой OpenCV нужно установить дополнительные пакеты, включая пакеты для работы с изображениями, видео и т. д. [13]:

- **cmake** — утилита для автоматической сборки программы из исходного кода;
- **cmake-curses-gui** — пакет GUI (графический интерфейс) для cmake;
- **libgtk2.0-dev** — пакет для создания графических пользовательских интерфейсов;
- **libjpeg-dev, libpng12-dev, libtiff5-dev** — библиотеки для работы с разными форматами изображений;
- **libjasper-dev** — JasPer — набор программ для кодирования и обработки изображений;
- **libavcodec-dev** — библиотека кодеков от Libav (аудио/видео);
- **libavformat-dev** — библиотека кодеков от Libav (аудио/видео);
- **libswscale-dev** — библиотека для выполнения высокооптимизированных масштабирований изображений;
- **libv4l-dev** — набор библиотек для работы с устройствами video4linux2;
- **libx264-dev** — библиотека кодирования для создания видеопотоков H.264 (MPEG-4 AVC);
- **libxvidcore-dev** — видеокодек MPEG-4 (Xvid);
- **gfortran** — матричные преобразования;
- **libatlas-base-dev** — библиотека процедур линейной алгебры.

Для этого также по очереди в строке терминала вводятся на исполнение следующие четыре команды:

```
sudo apt-get install cmake cmake-curses-gui  
libgtk2.0-dev
```

```
sudo apt-get install libavcodec-dev libavformat-dev  
libswscale-dev libv4l-dev libx264-dev libxvidcore-dev
```

```
sudo apt-get install libjpeg-dev libpng12-dev  
libtiff5-dev libjasper-dev
```

```
sudo apt-get install gfortran libatlas-base-dev
```

Шаг 3. Кроме того, полезно будет установить дополнительные пакеты, поддерживающие работу с видео под различными кодеками:

```
sudo apt-get install -y libdc1394-22-dev  
libavresample-dev libgphoto2-dev ffmpeg libgtk-3-dev
```

Затем идет установка библиотеки `numpy`, которая может занимать длительное время:

```
sudo pip3 install numpy
```

Шаг 4. Теперь на очереди скачивание и распаковка исходных файлов OpenCV. Для этого создадим с помощью команд в терминале рабочую папку с именем **opencv** в папке **/home/pi** и перейдем в нее:

```
cd /home/pi  
mkdir opencv  
cd opencv
```

Чтобы скачать и распаковать последние версии архивов с исходниками библиотеки¹ [11] и дополнительных модулей² [12], в терминале необходимо ввести:

¹ Open Source Computer Vision Library // GitHub. URL: <https://github.com/opencv/opencv> (Дата обращения 15.04.2019).

² Repository for OpenCV's extra modules // GitHub. URL: https://github.com/opencv/opencv_contrib (Дата обращения 15.04.2019).

```
wget-O opencv.zip https://github.com/opencv/opencv/
archive/master.zip
```

```
unzip opencv.zip
```

```
wget-O opencv_contrib.zip https://github.com/opencv/
opencv_contrib/archive/master.zip
```

```
unzip opencv_contrib.zip
```

Рекомендуется использовать одинаковые релизные версии библиотеки и дополнительных модулей, чтобы избежать появления различных проблем с установкой. С этой целью исходники можно скачать и распаковать с помощью следующих команд:

```
wget https://github.com/opencv/opencv/
archive/3.4.0.zip -O opencv_source.zip
```

```
unzip opencv_source.zip
```

```
wget https://github.com/opencv/opencv_contrib/
archive/3.4.0.zip -O opencv_contrib.zip
```

```
unzip opencv_contrib.zip
```

Шаг 5. Прежде чем приступить к сборке пакета OpenCV, необходимо создать отдельную папку для файла конфигурации:

```
cd /home/pi/opencv/opencv34
mkdir build
cd build
```

Шаг 6. Далее необходимо настроить параметры сборки при помощи утилиты **cmake**, которая была установлена ранее:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D INSTALL_C_EXAMPLES=OFF \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D OPENCV_EXTRA_MODULES_PATH=/home/pi/opencv/
opencv_contrib-3.4.0/modules \
```

```
-D BUILD_EXAMPLES=ON \  
-D BUILD_DOCS=ON \  
-D ENABLE_NEON=ON ..
```

Создание конфигурации занимает довольно продолжительное время. Признаком успешного завершения конфигурирования будет вывод на экран сообщений, помеченных на **рис. 3.6**.



```
pi@raspberrypi:~/opencv/opencv-3.4.0/build  
File Edit Tabs Help  
-- Install to: /usr/local  
-----  
-- Configuring done  
-- Generating done  
-- Build files have been written to: /home/pi/opencv/opencv-3.4.0/build  
pi@raspberrypi:~/opencv/opencv-3.4.0/build $
```

Рис. 3.6. Сообщение об успешном создании конфигурации параметров сборки OpenCV

Шаг 7. При успешном завершении конфигурации можно приступить к сборке с помощью команды **make -jn** (где параметр **-jn** задает количество задействованных в сборке ядер процессора Raspberry Pi, максимум 4):

```
make -j4
```

Успешное завершение компиляции подтвердится выводом сообщений на экран (**рис. 3.7**).

Процесс компиляции очень длительный, может занять несколько часов. При этом процессор Raspberry Pi ощутимо греется, что в случае неэффективного охлаждения может привести к зависанию системы. Несмотря на предоставленную возможность задействовать все четыре ядра в погоне за скоростью, для компиляции рекомендуется все-таки задействовать меньшее количество ядер (1 или 2), используя ключи **-j1** или **-j2**. Это обеспечит более щадящий и более стабильный режим работы. Если вы все же отважились задействовать все ядра и это привело к зависанию, следует после перезагрузки возобновить процедуру компиляции с меньшим числом ядер, не забывая при этом перейти в ранее созданную папку **build**:

```
cd /home/pi/opencv/opencv34/build  
make -j2
```

```

pi@raspberrypi:~/opencv/opencv-3.4.0/build
File Edit Tabs Help
[ 97%] Built target tutorial_CannyDetector_Demo
Scanning dependencies of target tutorial_Geometric_Transforms_Demo
[ 97%] Building CXX object samples/cpp/CMakeFiles/tutorial_Geometric_Transforms_Demo.dir/tutorial_code/ImgTrans/Geometric_Transforms_Demo.cpp.o
[ 97%] Linking CXX executable ../../bin/cpp-tutorial-Geometric_Transforms_Demo
[ 97%] Built target tutorial_Geometric_Transforms_Demo
Scanning dependencies of target tutorial_HoughCircle_Demo
[ 97%] Building CXX object samples/cpp/CMakeFiles/tutorial_HoughCircle_Demo.dir/tutorial_code/ImgTrans/HoughCircle_Demo.cpp.o
[ 98%] Linking CXX executable ../../bin/cpp-tutorial-HoughCircle_Demo
[ 98%] Built target tutorial_HoughCircle_Demo
[ 98%] Built target example_dnn_resnet_ssd_face
[ 98%] Built target example_dnn_fc7_semsegm
[ 98%] Built target example_dnn_squeezenet_halide
[ 98%] Built target example_dnn_caffe_googlenet
[ 98%] Built target example_dnn_torch_enet
[ 99%] Built target example_dnn_ssd_mobilenet_object_detection
[ 99%] Built target example_dnn_ssd_object_detection
[ 99%] Built target example_dnn_faster_rcnn
[ 99%] Built target example_dnn_tf_inception
[ 99%] Built target example_dnn_yolo_object_detection
[ 99%] Built target example_tapi_clahe
[ 99%] Built target example_tapi_pyrik_optical_flow
[ 99%] Built target example_tapi_bgfg_segmm
[ 99%] Built target example_tapi_camshift
[100%] Built target example_tapi_tvll_optical_flow
[100%] Built target example_tapi_hog
[100%] Built target example_tapi_opencv_custom_kernel
[100%] Built target example_tapi_squares
[100%] Built target example_tapi_ufacedetect
pi@raspberrypi:~/opencv/opencv-3.4.0/build $

```

Рис. 3.7. Вывод сообщений об успешном завершении компиляции

Шаг 8. В завершение после компиляции остается проинсталлировать пакет OpenCV и сформировать необходимые связи и кэш. Делается это с помощью следующих команд:

```

sudo make install
sudo ldconfig

```

Шаг 9. Для проверки того, что библиотека OpenCV установлена правильно, в оболочке Shell (Python) вводим построчно команды (**рис. 3.8.**):

```

>>> import cv2
>>> cv2.__version__

```

Более подробно с процедурой установки OpenCV на Raspberry Pi 3 можно ознакомиться в [10, 13, 14].

```

Python 3.5.3 Shell
File Edit Shell Debug Options Window Help
Python 3.5.3 (default, Jan 29 2019, 17:19:29)
[GCC 6.3.0 20170124] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import cv2
>>> cv2.__version__
'3.4.0'
>>> |
Ln: 7 Col: 4

```

Рис. 3.8. Проверка установки библиотеки OpenCV

Удаленное управление Raspberry Pi 3

Итак, мы установили все необходимые библиотеки, можно начинать программировать — пора «дать зрение» нашему Raspberry Pi 3! Программировать, конечно, можно непосредственно на Raspberry Pi 3, но на практике это не совсем удобно, особенно если он управляет подвижным роботом. В этом случае в процессе отладки наших программ мы получим сомнительное удовольствие, когда за роботом будет тянуться шлейф из проводов от монитора, клавиатуры и мыши. Поэтому работать с Raspberry Pi 3 мы будем удаленно. Удаленный доступ к ресурсам Raspberry Pi 3 можно реализовать различными способами, они достаточно подробно описаны в Интернете^{1, 2} [15, 16].

Рассмотрим один из способов удаленной работы с рабочим столом Raspbian. При стандартном варианте установки Raspbian подключиться к Raspberry Pi 3 можно через интерфейс VNC Server.

ПОЯСНЕНИЕ

VNC (Virtual Network Computing) — платформонезависимая система удаленного доступа к рабочему столу компьютера, осуществляющая управление путем передачи нажатий клавиш на клавиатуре и движений мыши с одного компьютера на другой и ретрансляции содержимого экрана через компьютерную сеть. Система VNC состоит из двух основных частей. Первая часть — VNC-клиент (VNC viewer), с его помощью осуществляется подключение и удаленное управление. Вторая — VNC-сервер — обеспечивает доступ к управляемым ресурсам. К одному VNC-серверу могут подключаться несколько клиентов одновременно.

Этот способ удаленного доступа позволяет полностью управлять и взаимодействовать с системой через ее графический интерфейс. Начиная с версии Raspbian 4.8, VNC Server включен в состав операционной системы.

¹ Способы удаленного управления Raspberry Pi 3. // Портал о микроконтроллере Raspberry Pi. URL: <https://myraspberrry.ru/sposobyi-udalennogo-upravleniya-raspberry-pi-3.html> (Дата обращения 15.04.2019).

² Raspberry Pi 3 и SAMBA: удаленный доступ к файлам Малинки по локальной сети. // CODIUS. URL: https://codius.ru/articles/Raspberry_Pi_3_и_SAMBA_удаленный_доступ_к_файлам_по_локальной_сети (Дата обращения 15.04.2019).

Еще нам потребуется постоянное проводное или беспроводное подключение Raspberry Pi 3 к сети Интернет. Лучше беспроводной вариант, например через Wi-Fi роутер. Если VNC Server по какой-либо причине отсутствует в операционной системе, его можно установить, набрав в терминале команду:

```
sudo apt-get install realvnc-vnc-server
```

Затем первое, что нужно сделать, — это проверить состояние VNC Server через утилиту, которую можно запустить через терминал, введя строку:

```
raspi-config
```

Появится окно настройки конфигурации Raspberry Pi — Raspberry Pi Configuration. Открываем вкладку *Interfaces* и ищем строку VNC. Необходимо выставить в значении данного параметра состояние Enabled (**рис. 3.9**). Этим самым автоматически будет запущен VNC Server. И... все! Настройку серверной части мы закончили.

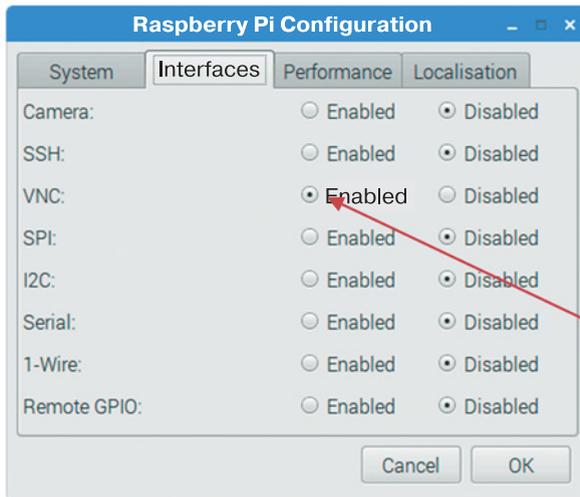


Рис. 3.9. Настройка конфигурации Raspberry Pi

Переходим ко «Второй части Марлезонского балета¹». Необходимо установить и настроить на рабочем компьютере **клиентскую часть** —

¹ А. Дюма. «Три мушкетера».

VNC Viewer. Рабочий компьютер тоже должен иметь подключение с активным доступом к Интернету. Следующий шаг — по ссылке¹ [17] скачиваем дистрибутив VNC Viewer под операционную систему рабочего компьютера (в нашем случае это будет Windows). Затем запускаем процесс установки, следуя указаниям мастера. После окончания установки клиента остается запустить VNC Viewer и настроить соединение с Raspberry Pi 3.

Выбирая в меню *File* → *New connection*, создаем новое соединение (рис. 3.10):

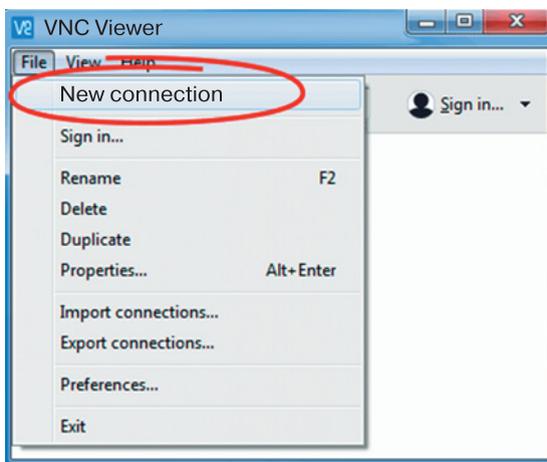


Рис. 3.10. Создание нового соединения VNC

В появившемся окне **Properties** нового соединения на вкладке *General* нужно указать IP-адрес VNC Server (это и есть наш Raspberry Pi 3), к которому будет установлено подключение (рис. 3.11). Через двоеточие указываем номер дисплея, заданный при запуске VNC-сервера (по умолчанию значение равно 0, его можно не указывать). IP-адрес Raspberry Pi 3 можно узнать, набрав в терминале Raspbian команду:

```
hostname -I
```

В поле **Name** вводим с клавиатуры имя соединения, например RaspPi3 (см. рис. 3.11):

¹ VNC® Connect consists of VNC® Viewer and VNC® Server // Realvnc. URL: <https://www.realvnc.com/en/connect/download/viewer/windows/> (Дата обращения 15.04.2019).

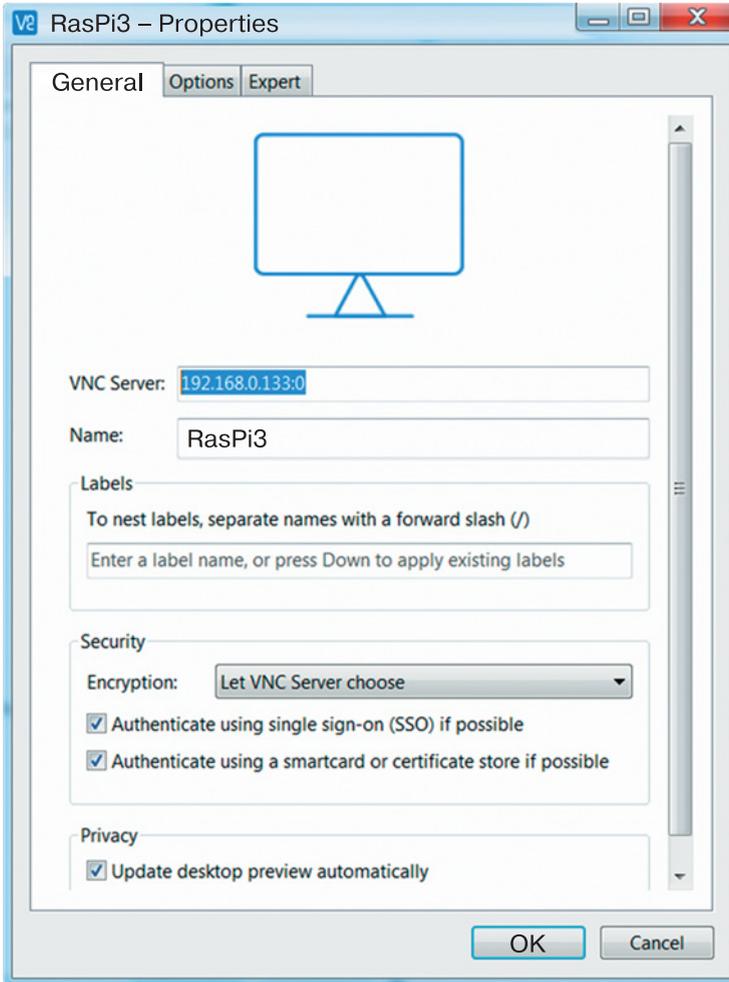


Рис. 3.11. Окно настроек соединения с Raspberry Pi 3

После нажатия кнопки **OK** в главном окне VNC Viewer будет создано новое соединение (**рис. 3.12**).

Чтобы подключить рабочий компьютер к Raspberry, надо сделать двойной клик на значке только что созданного соединения **RasPi3** (см. **рис. 3.12**). Если все сделали правильно, то появится окно авторизации (**рис. 3.13**).

Далее вводится имя пользователя и пароль. По умолчанию имя пользователя: **pi**, пароль: **raspberry**. После нажатия **OK** на экране рабочего компьютера появится окно с рабочим столом Raspbian (см. **рис. 3.5**).

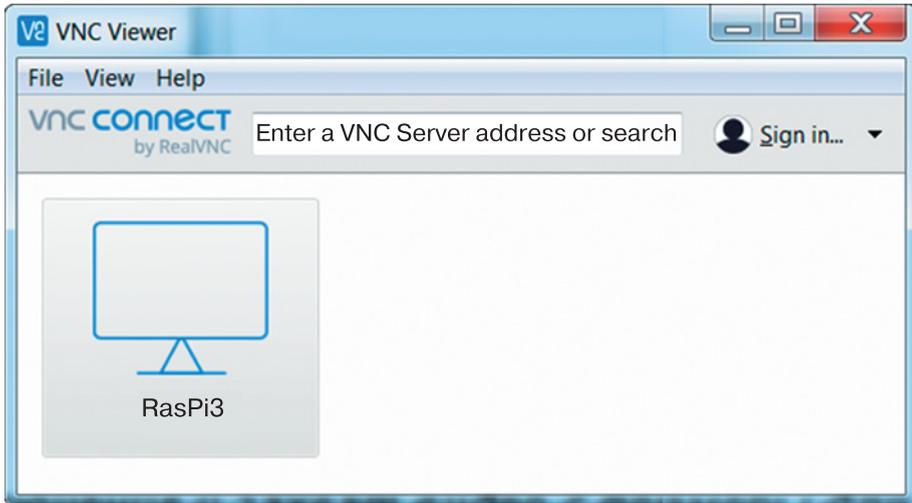


Рис. 3.12. Главное окно VNC Viewer с новым соединением

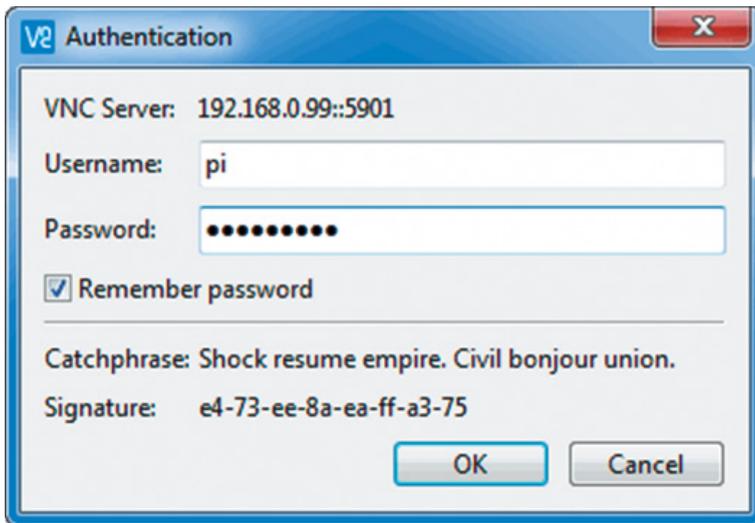


Рис. 3.13. Окно авторизации с Raspberry Pi3

Подключение модулем камеры CSI для Raspberry Pi и захват видео

В предыдущих темах мы уже изучали захват и обработку видео средствами OpenCV с web-камеры и из видеофайлов. В этом разделе будут рассмотрены подключение и захват видео с помощью специального модуля камеры CSI для Raspberry Pi (рис. 3.14):

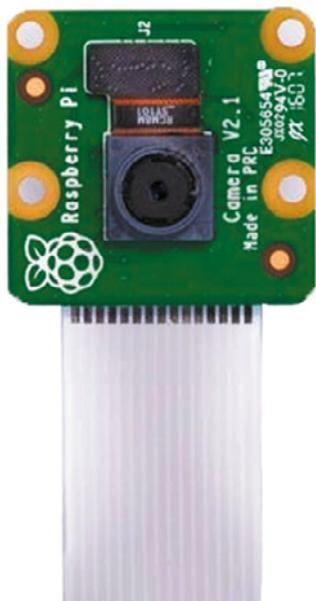


Рис. 3.14. Модуль камеры CSI для Raspberry Pi

Подключение камеры к Raspberry Pi не представляет особых сложностей. Для этого надо воспользоваться специальным шлейфом (см. рис. 3.14). Прежде чем приступить к этой тонкой работе, обязательно убедитесь, что питание на Raspberry Pi **отключено!** Далее найдите на плате Raspberry Pi порт для подключения камеры (см. рис. 3.1) и аккуратно (нежно), **не прилагая больших усилий**, вытяните фиксатор шлейфа вверх за боковые выступы. При этом верхняя часть фиксатора поднимется примерно на 2 мм и отойдет немного в сторону. После этого свободный конец шлейфа опустите в открывшийся разъем так, чтобы синяя метка была со стороны аудиоразъема, а контакты были направлены на HDMI-разъем. Затем так же, **не демонстрируя молодецкой удали**, аккуратно прижмите фиксатор к разъему для камеры. После подключения все должно выглядеть, как на **рис. 3.15**, без лишних «запасных частей» и мелких фрагментов пластика на поверхности стола.



Рис. 3.15. Подключение шлейфа CSI-камеры к разъему платы Raspberry Pi (Источник: <https://robx.org/wp-content/uploads/2019/07/connect-camera.jpg>)

Теперь проверим работу CSI-камеры. Для начала ее нужно включить. Заходим в терминал и с помощью команды

```
raspi-config
```

открываем уже знакомое нам окно Raspberry Pi Configuration (см. рис. 3.9). В разделе *Interfaces* находим строку Camera и включаем состояние *Enabled*. Аналогичные действия мы проделали, когда запускали VNC Server. Теперь можно получить доступ к изображению со CSI-камеры.

К этому времени вы уже должны были, не сдерживая свой неуемный интерес ко всему новому, исследовать меню программ (кнопка в левом верхнем углу экрана на рис. 3.5) и найти там Python. Если этого не случилось, находим его там, открываем. Затем вводим шесть строчек программного кода, который представлен ниже, и запускаем:

```
from picamera import PiCamera
from time import sleep
camera = PiCamera()
camera.start_preview()
sleep(15)
camera.stop_preview()
```

На экране компьютера должно появиться окно, передающее изображение с камеры, и через 15 с закрыться. Надо заметить, что здесь для работы со CSI-камерой применяется специальная библиотека `picamera`.

ПОЯСНЕНИЕ

`picamera` — это пакет для Python 2.7 и выше, представляющий собой прикладной программный интерфейс (API — Application Programming Interface) для работы с модулем камеры CSI для Raspberry Pi. С оригинальной документацией по работе с этим пакетом можно ознакомиться здесь¹.

Убедившись, что камера дает изображение, рассмотрим возможность захвата видео средствами OpenCV и сохранения видеоданных в файл `testvideo.avi`. Кадры с камеры будут выводиться в окно `Frames`. Для этого нам потребуется импортировать классы `PiRGBArray` и `PiCamera` и понять, как они работают.

`PiRGBArray` — это пользовательский выходной класс, определенный в модуле `picamera.array`. Этот класс создает трехмерный массив (строки, столбцы, цвета) для захвата RGB-изображения. Сам модуль `picamera.array` представляет собой прикладной программный интерфейс и содержит набор классов, предназначенных для работы с n -мерными numpy-массивами выходных данных камеры. Модуль `picamera.array` не импортируется автоматически с основным пакетом `picamera` и должен быть явно импортирован.

`PiCamera` — это класс из модуля `picamera.camera`, является основным интерфейсом для работы со CSI-камерой Raspberry. Все классы из модуля `picamera.camera` доступны из пространства имен самого пакета `picamera`. Поэтому их можно импорти-

¹ Picamera 1.13 Documentation. // URL: <https://picamera.readthedocs.io/en/release-1.13/index.html> (Дата обращения 10.01.2020)

ровать без непосредственного импорта модуля `picamera.camera`. После создания этот класс инициализирует камеру. Нам надо будет указать параметры: разрешение (**resolution**) и частоту смены кадров (**framerate**). Остальные параметры оставим со значениями по умолчанию.

После того как мы получили доступ к данным видеопотока со CSI-камеры, для их записи в файл воспользуемся инструментами OpenCV. Таким инструментом будет функция `cv2.VideoWriter`. В общем виде функция имеет следующее представление:

```
cv2.VideoWriter(fileout, FourCC, fps, resolution,  
                isColor)
```

где аргументы имеют следующее значение:

fileout — имя выходного файла, например **output.avi**;

FourCC — 4-байтовый код идентификации видекодека, используемого для компрессии видео (список доступных кодов можно найти на сайте¹, работа кодека зависит от операционной платформы²);

fps — число кадров в секунду;

resolution — размер кадра, например (640 × 480);

isColor — флаг, по умолчанию равен True — ожидание цветного кадра, в противном случае — оттенки серого кадра.

Параметр **FourCC** передается посредством вызова инструкции `cv2.VideoWriter_fourcc(*'XVID')`, где **XVID** — символьная последовательность обозначения кодека (см. здесь³).

Давайте опишем алгоритм работы программы. Он будет состоять из следующих действий.

1. Инициализация камеры.
2. Подготовка файла для записи видеоданных.
3. Запуск цикла по кадрам видеопотока.
4. Покадровое чтение данных видеопотока камеры.
5. Покадровая запись данных видеопотока в файл.
6. Прерывание работы цикла.
7. Завершение работы программы.

¹ FourCC (ресурс о видекодеках и форматах пикселей). // URL: <http://fourcc.org/codecs.php> (Дата обращения 10.01.2020).

² Open CV. Getting Started with Videos. // URL: https://docs.opencv.org/master/dd/d43/tutorial_py_video_display.html (Дата обращения 10.01.2020).

³ FourCC (ресурс о видекодеках и форматах пикселей). // URL: <http://fourcc.org/codecs.php> (Дата обращения 10.01.2020).

А теперь разберемся с программным кодом, пример которого приводится ниже:

```
# импортируем класс для захвата RGB-массива
from picamera.array import PiRGBArray
# импортируем класс PiCamera
from picamera import PiCamera
# импортируем модуль работы со временем
import time
# импортируем пакет OpenCV
import cv2
# инициализируем камеру,
# создаем экземпляр класса PiCamera
camera = PiCamera()
# устанавливаем размер кадра
camera.resolution = (640, 480)
# задаем частоту смены кадров в секунду
camera.framerate = 24
# создаем трехмерный массив RGB
raw = PiRGBArray(camera, size=(640, 480))
# ожидание для прогрева матрицы камеры
time.sleep(0.1)
# подготовка к записи в файл,
# получаем идентификатор видеокodeка
fourcc = cv2.VideoWriter_fourcc(*'XVID')
# создаем объект в переменной out для записи видео
# в файл
out = cv2.VideoWriter('testvideo.avi', fourcc, 20.0,
                                               (640, 480))
# запускаем покадровый циклический процесс
# чтения, отображения и записи в файл данных
# видеопотока
for frame in camera.capture_continuous(raw,
                                                                       format="bgr",
                                                                       use_video_port=True):
# читаем в image массив данных текущего захваченного
# кадра
image = frame.array
# отображаем в окне Frames изображение текущего кадра
cv2.imshow("Frames", image)
# передаем текущий кадр для записи ранее созданному
```

```
# объекту out
out.write(image)
# очищаем RGB-массив для чтения следующего кадра
raw.truncate(0)
# ожидаем нажатия клавиши "q"
if cv2.waitKey(1) & 0xFF == ord("q"):
# если да, прерываем цикл
break
# после работы освобождаем память
camera.close()
out.release()
# и закрываем все окна
cv2.destroyAllWindows()
```

Как вы уже заметили, для организации циклического процесса работы с кадрами мы использовали метод `camera.capture_continuous()`. Этот метод представляет собой бесконечный итератор изображений, полученных непрерывно с камеры. Его работа аналогична методу `camera.capture`¹, только он возвращает объекты изображения последовательно, кадр за кадром, до тех пор, пока его работа не будет прервана. Для захвата видео методу `camera.capture_continuous()` в качестве аргументов передаем объект класса `PiRGBArray` (трехмерный RGB-массив), формат пикселя `format` (в нашем случае это `bgr`, кодирующий цвет 24 битами) и используемый порт вывода изображения `use_video_port=True`. Надо сказать, что модуль CSI-камеры для Raspberry Pi имеет два разных порта для вывода изображения: видеопорт и фотопорт. Видеопорт более производительен, но качество изображения при этом не на высоте (для его включения передается значение `True`). Фотопорт всегда задействован по умолчанию (значение `False`) и работает намного медленнее, зато выдает изображения высокого качества.

Внутри цикла каждый кадр записывается в итерационную переменную `frame`, к которой применяется метод `array`. С помощью этого метода мы преобразуем кадры в массивы, доступные для инструментов OpenCV. Как видите, программировать в Python на Raspberry не так уж и страшно!

¹ Picamera 1.13 Documentation. API — The PiCamera Class // URL: https://picamera.readthedocs.io/en/release-1.13/api_camera.html#picamera (Дата обращения 10.01.2020).

Управление моторами с помощью Raspberry Pi на примере робота AlphaBot

В прошлом разделе мы «открыли глаза» нашему Raspberry, осталось «приделать ноги и научить его ходить»! Другими словами, чтобы обеспечить Raspberry Pi мобильными функциями, требуется подобрать или создать для него подвижную платформу. Здесь, конечно, можно рассмотреть много вариантов и сравнений в пользу тех или иных решений. Но мы, для примера, воспользуемся одним из самых доступных вариантов. Тем более что это просто инструмент для достижения нашей основной цели — освоения технологии компьютерного зрения.

Итак, в качестве мобильной платформы для изучения роботизированных систем, оснащенных компьютерным зрением, выберем роботоплатформу AlphaBot [18] от Waveshare Electronics (**рис. 3.16**).

Для тех, кто не знаком с AlphaBot, — буквально пару слов, заодно немного порадует разработчика этой платформы.

AlphaBot была специально разработана для решения задач компьютерного зрения с помощью мобильного робота. Платформа имеет посадочные места для крепления платы Raspberry Pi 3 и модуля камеры CSI. Поскольку корпус представляет собой по совместительству еще и плату, то все электрические компоненты и основные датчики уже интегрированы. Это сводит к минимуму монтажные работы и исключает паечные. Самое же главное достоинство — отсутствие мешанины проводов, а значит, и ошибок с их подключением или об-

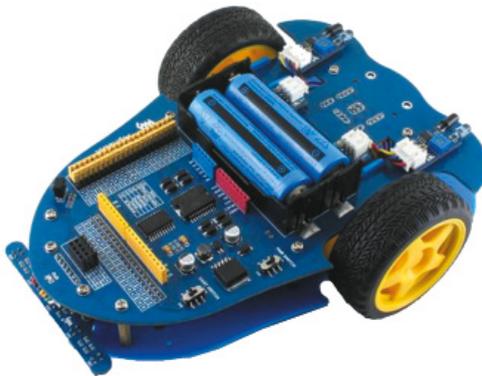


Рис. 3.16. Робоплатформа AlphaBot (Источник: <https://www.waveshare.com/product/robotics/alphabot/platforms.htm>)

рывом соединения из-за выпавшего штекера. Таким образом, после сборки шасси AlphaBot и установки на нем Raspberry Pi 3 можно сразу приступить к программированию нашего робота. Ах да! Предварительно надо зарядить аккумуляторы.

А теперь к серьезным вещам! Прежде чем решать задачу управления мобильной платформой посредством данных, получаемых с видеокамеры, выясним принципы управления ходовыми моторами. Для реализации управления ходовыми моторами на плате самого робота по умолчанию выполнена разводка, связывающая драйверы моторов с цифровыми портами GPIO¹ под номерами 12, 13, 20, 21, 6, 26.

ПОЯСНЕНИЕ

Цифровые порты № 12 и 13 отвечают за направление вращения левого мотора, а цифровые порты № 20 и 21 — за направление вращения правого мотора. Цифровые порты № 6 и 26 являются разрешающими контактами. После подачи разрешающего уровня сигнала с помощью данных портов определяется скорость вращения моторов. С режимами включения моторов можно ознакомиться в таблице ниже:

Уровень сигнала цифровых портов: 1 — HIGH (3,3 В), 0 — LOW (0 В)				Направление вращения моторов/Движение робота
№ 12	№ 13	№ 20	№ 21	
1	0	0	1	Моторы вращаются вперед/ Робот идет прямо вперед
0	1	1	0	Моторы вращаются назад/ Робот идет прямо назад
0	0	0	1	Левый — стоп, правый — вперед/ Поворот влево
1	0	0	0	Левый — вперед, правый — стоп/ Поворот вправо
0	0	0	0	Моторы — стоп/Робот — стоп

¹ AlphaBot User Manual // URL: <https://www.waveshare.com/w/upload/c/c7/AlphaBot-User-Manual.pdf> (Дата обращения 10.01.2020).

С портами GPIO мы уже «заочно» познакомились: узнали, что это и где они расположены, теперь настало время подружиться с ними. Поможет нам в этом библиотека **RPi.GPIO**.

ПОЯСНЕНИЕ

RPi.GPIO — библиотека на языке Python, которая применяется для управления портами GPIO в микрокомпьютере Raspberry Pi. Основные функции библиотеки **RPi.GPIO**:

- 1) считывание состояния портов, сконфигурированных на вход (input);
- 2) реакция на прерывания, инициируемые любым из портов в режиме input;
- 3) управление состоянием портов, сконфигурированных на выход (output);
- 4) генерация **PWM (Pulse-Width Modulation)** — широтно-импульсной модуляции (ШИМ) на портах в режиме output;
- 5) получение информации о платформе и конфигурации портов.

Более подробно о работе с GPIO можно ознакомиться на этих ресурсах в Интернете^{1, 2, 3, 4}.

С помощью библиотеки **RPi.GPIO** мы выполним настройку перечисленных выше портов GPIO, на которые будем подавать команды для управления вращением моторов. Попытаемся разобраться в этом вопросе. Для начала необходимо инициализировать GPIO. В этом нам помогут два метода: **GPIO.setmode()** и **GPIO.setup()**. Методом **GPIO.setmode()** определяется режим нумерации контактной группы (портов). Здесь возможны два режима: **BCM** и **BOARD** (рис. 3.17). Для нашего случая выбираем режим **BCM**:

GPIO.setmode (BCM)

¹ RPi.GPIO // URL: <https://pypi.org/project/RPi.GPIO/> (Дата обращения 10.01.2020).

² Raspberry Pi Documentation. GPIO // URL: <https://www.raspberrypi.org/documentation/usage/gpio/> (Дата обращения 10.01.2020).

³ Ph0en1x.net.RPi.GPIO — работа с входами, выходами и прерываниями в Raspberry Pi, примеры на Python // URL: <https://ph0en1x.net/106-rpi-gpio-installation-working-with-inputs-outputs-interrupts.html> (Дата обращения 10.01.2020).

⁴ CODIUS. Raspberry Pi 3: GPIO // URL: https://codius.ru/articles/Raspberry_Pi_3_GPIO_введение (Дата обращения 10.01.2020).

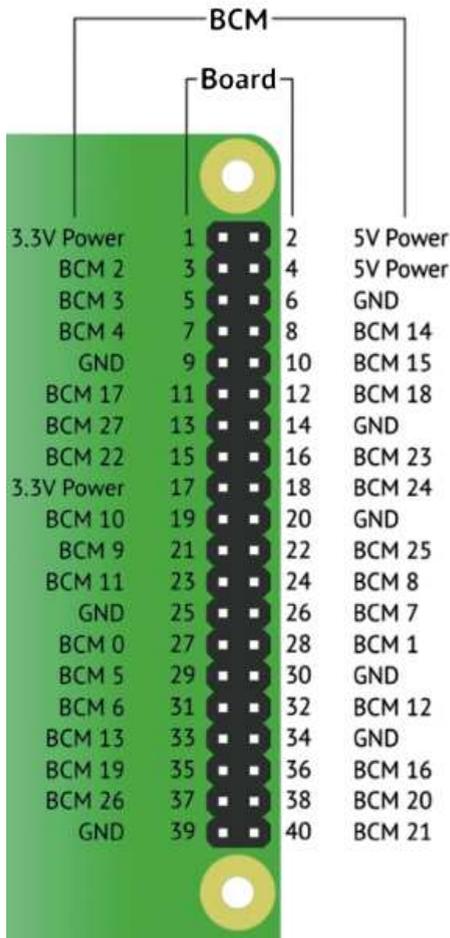


Рис. 3.17. Нумерация пинов GPIO в режимах BCM и BOARD

Чтобы иметь возможность считывать и устанавливать уровни сигналов на портах, нужно выполнить настройку их конфигурации. Настройка конфигурации осуществляется вызовом метода `GPIO.setup()`. Например, в следующем фрагменте кода цифровой порт № 7 будет работать как выход, а порт № 22 — как вход.

```
GPIO.setup(7, GPIO.OUT)
GPIO.setup(22, GPIO.IN)
```

Параметры `GPIO.OUT` и `GPIO.IN` являются константами модуля `RPi.GPIO`, значения которых равны «0» и «1» соответственно. Для

удобочитаемости кода при написании программ предпочтительно указывать имена констант, а не их значения.

На цифровых портах Raspberry уровень сигнала (напряжения) может принимать только два значения: HIGH и LOW («1» и «0»). Как же управлять скоростью вращения, когда нет промежуточных уровней? Чтобы осуществлять плавное управление мотором, на управляющем порте генерируется ШИМ-сигнал.

ПОЯСНЕНИЕ

Широтно-импульсная модуляция (ШИМ, англ. PWM — pulse-width modulation) применяется для плавного изменения мощности, поступающей на нагрузку от источника питания. Например, с целью регулирования скорости вращения вала двигателя, плавности изменения яркости освещения или подсветки.

При ШИМ напряжение питания подается на нагрузку не постоянно, а прерывисто — импульсами. Импульсы напряжения имеют постоянную амплитуду и постоянный период следования, а вот ширина их изменяется. Чем большую мощность нужно подать на нагрузку, тем большей делается ширина импульсов (**рис. 3.18**).

Величина сигнала ШИМ характеризуется **коэффициентом заполнения D** :

$$D = t_n/T,$$

где t_n — ширина импульсов, а T — период их следования.

Величина, обратная коэффициенту заполнения, называется **скважностью S** . На практике удобнее пользоваться коэффициентом заполнения.

В Raspberry на всех портах GPIO доступна программная реализация ШИМ¹ с помощью библиотеки RPi.GPIO (начиная с версии 0.5.2a; на момент написания книги актуальная версия 0.7.0²). Чтобы выдать ШИМ-сигнал на каком-то порте, нужно с помощью функции **GPIO.PWM()** создать объект его управления. При создании объекта в аргументе функции указываются номер порта и частота сигнала ШИМ. Максимально возможная частота сигнала ШИМ составляет 8,5 кГц. Этот объект имеет три интересных для нас метода:

¹ На микрокомпьютере Raspberry имеется только один порт GPIO № 12 (BCM № 18) с аппаратной поддержкой ШИМ.

² RPi.GPIO. // URL: <https://pypi.org/project/RPi.GPIO/> (Дата обращения 10.01.2020).

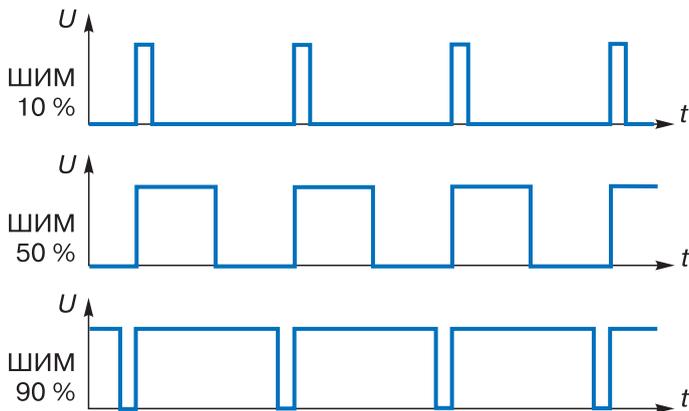


Рис. 3.18. Сигнал ШИМ с коэффициентом заполнения 10%, 50%, 90%

- 1) метод **start()** запускает ШИМ-сигнал с указанным коэффициентом заполнения (0–100%);
- 2) метод **ChangeDutyCycle()** позволяет изменить установленный коэффициент заполнения сигнала на указанное в аргументе значение;
- 3) метод **ChangeFrequency()** позволяет изменить частоту (период следования импульсов) сигнала ШИМ; в аргументе указывается новое значение частоты сигнала ШИМ.

Чтобы установить уровень сигнала выходного пина (контакта), нужно применить метод **GPIO.output()**. В аргументах данного метода задается номер цифрового порта и уровень сигнала: **GPIO.HIGH** или **GPIO.LOW**. Теперь мы почти готовы, чтобы оживить нашу робоплатформу. Для начала пусть она просто проедет немного вперед. Рассмотрим пример небольшого программного кода первого маневра AlphaBot:

```
# импортируем модуль GPIO
import RPi.GPIO as GPIO
# импортируем модуль time
import time
# запишем в константы разъемы управления моторами
# AlphaBot
# левый мотор
PN1 = 12
PN2 = 13
ENA = 6
```

```
# правый мотор
PN3 = 20
PN4 = 21
ENB = 26
# инициализация GPIO
# задаем режим BCM для нумерации портов GPIO
GPIO.setmode(GPIO.BCM)
# задаем конфигурацию цифровых портов
GPIO.setup(PN1, GPIO.OUT)
GPIO.setup(PN2, GPIO.OUT)
GPIO.setup(PN3, GPIO.OUT)
GPIO.setup(PN4, GPIO.OUT)
GPIO.setup(ENA, GPIO.OUT)
GPIO.setup(ENB, GPIO.OUT)
# создаем ШИМ-объекты для плавного управления
# скоростью вращения моторами с частотой сигнала
# 100 Гц
PWMA = GPIO.PWM(ENA, 100)
PWMB = GPIO.PWM(ENB, 100)
# запускаем ШИМ = 30%, то есть включаем моторы
# на 30% мощности,
# здесь можно управлять скоростью вращения моторов
PWMA.start(30)
PWMB.start(30)
# НАЧАЛО МАНЕВРА
# задаем движение вперед прямо:
# левый мотор,
GPIO.output(PN1, GPIO.HIGH)
GPIO.output(PN2, GPIO.LOW)
# правый мотор,
GPIO.output(PN3, GPIO.LOW)
GPIO.output(PN4, GPIO.HIGH)
# 5 секунд - полет нормальный! Не переходим к другим
# командам в течение указанного времени в секундах
time.sleep(5)
# остановка моторов
PWMA.stop()
PWMB.stop()
# КОНЕЦ МАНЕВРА
# освобождаем ресурсы GPIO
GPIO.cleanup()
```

Как видите, не так уж и сложно запрограммировать нашего робота прокатиться по прямой. Но если вдруг робот закрутился или поехал назад, поменяйте местами провода моторов, которые вращаются неправильно. А меняя значения уровней выходных сигналов в методе `GPIO.output()` для портов `PN1–PN4`, можно программно изменить направление движения робота.

ЗАДАНИЯ

1. Составьте программу для выполнения роботом AlphaBot простейших маневров: поворотов на 90° вправо и влево и разворот, подбирая время включения моторов. Значения уровней сигнала портов возьмите из таблицы на с. 143.
2. Составьте программу для движения робоплатформы по траектории в виде треугольника, прямоугольника и окружности, последовательно соединяя фрагменты программ для простейших маневров.

Программирование движения робота AlphaBot вдоль черной линии

И вот мы пришли к задаче, в которой надо запрограммировать робота двигаться по траектории в соответствии с данными, полученными с видеокamеры. Для этого сначала подготовим поле с черной линией на белой основе. У нас будет поле, файл для печати которого можно найти по ссылке¹. Рассмотрим задачу программирования робота AlphaBot с установленной CSI-камерой для движения вдоль черной линии. После установки камеры необходимо убедиться, что поверхность с черной линией попадает в поле зрения камеры.

В основу программы положим алгоритм, который называется пропорциональный регулятор. **Пропорциональный регулятор** — это автомат с обратной связью, в котором управляющее воздействие прямо пропорционально величине отклонения от нормы. И как мы это будем делать? А вот как!

¹ Портал программы «Робототехника» // Макеты полей для соревнований «Hello, Robot! Open». Сезон 2018–2019. URL: <http://russianrobotics.ru/upload/iblock/264/264860ce56de8c0057f5dce9c5d712f5.pdf> (Дата обращения: 10.01.2020).

Сначала определимся с нормой. Будем считать так. Если изображение черной линии проходит посередине кадра, это норма — едем прямо. Если черная линия находится в правой или левой области кадра, то мы совершаем маневр в правую или левую сторону, соответственно, пропорционально величине отклонения от середины. Однако линия имеет протяженный характер и могут встретиться криволинейные участки. А это усложнит задачу анализа изображения с камеры. Поэтому мы упростим себе жизнь: вырежем горизонтальную полоску кадра, в которой почти прямоугольный кусочек черной линии будет служить разделителем этой полоски условно пополам (**рис. 3.19**). Точнее, к этому условному местонахождению разделителя мы будем стремиться — это и будет нормой.

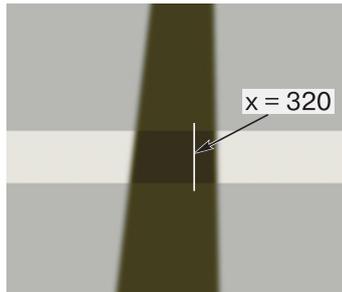


Рис. 3.19. Поле зрения видеокamеры робота, из которого вырезается горизонтальная область с изображением фрагмента черной линии на белом основании. Значение $x = 320$ — это средняя вертикаль кадра при разрешении 640×480

А теперь составим алгоритм программы движения робота вдоль черной линии. Это будет циклическая последовательность действий:

- Захват кадра с изображением черной линии.
- Вырезка горизонтального фрагмента изображения камеры в ширину кадра и высотой в 10–20 пикселей.
- Наложение цветовой RGB (BGR)-маски на вырезанный фрагмент для выделения пятна черной линии.
- Вычисление текущего положения центра пятна черной линии по горизонтальной оси.
- Расчет отклонения текущего положения центра пятна черной линии от геометрического центра фрагмента кадра по горизонтальной оси:

— если пятно слева, то расчет для поворота влево, поворот влево;

- если пятно справа, то расчет для поворота вправо, поворот вправо;
- если пятно посередине, то движение прямо.

В принципе алгоритм программы понятен, осталось разобрать его в коде на языке Python. Внимательно смотрим, как это будет выглядеть:

```
# импортируем класс для захвата RGB-массива
from picamera.array import PiRGBArray

# импортируем класс PiCamera
from picamera import PiCamera
# импортируем модуль работы со временем
import time
# импортируем библиотеку numpy
import numpy as np
# импортируем пакет OpenCV
import cv2
# импортируем модуль GPIO
import RPi.GPIO as GPIO
# запишем в константы порты управления моторами
# AlphaBot
# левый мотор
PN1 = 12
PN2 = 13
ENA = 6
# правый мотор
PN3 = 20
PN4 = 21
ENB = 26
# инициализация GPIO
# задаем режим BCM для нумерации портов GPIO
GPIO.setmode(GPIO.BCM)
# задаем конфигурацию портов
GPIO.setup(PN1, GPIO.OUT)
GPIO.setup(PN2, GPIO.OUT)
GPIO.setup(PN3, GPIO.OUT)
GPIO.setup(PN4, GPIO.OUT)
GPIO.setup(ENA, GPIO.OUT)
GPIO.setup(ENB, GPIO.OUT)
# создаем ШИМ, объекты для плавного управления
```



```
# читаем в image массив данных текущего захваченного
# кадра
    image = frame.array
# вырезаем полоску изображения, используя срезы
# x1:x2, y1:y2
    crop = image[150:180, 0:640]
# задаем диапазон маски для черного цвета
    low_color = (0,0,0)
    high_color = (110,110,110)
# накладываем маску черного цвета на вырезанный
# фрагмент
    only_object = cv2.inRange(crop, low_color,
                              high_color)
# рассчитываем моменты полученного белого пятна
# после фильтрации:
    moments = cv2.moments(only_object, 1)
    y_m = moments['m01']
    x_m = moments['m10']
    area = moments['m00']
# исключаем шум
    if area > 500:
# находим центр пятна черной линии
        x = int(x_m / area)
        y = int(y_m / area)
# выводим его координаты, после тестирования строки
# можно убрать
        cv2.circle(image, (x, y), 2, clr_lb, 1)
        cv2.putText(image, "%d-%d" % (x,y),
                    (x+10,y+30), cv2.FONT_HERSHEY_SIMPLEX,
                    0.5, clr_lb, 1)
# считаем абсолютное
# (разность между центром полоски экрана и центром
# пятна)
        er = x - x0
# и относительное отклонение (от -1 до 1)
        der = er/x0
# если сравнимо с нулем, едем прямо
        if er == 0:
# оба мотора крутим с одинаковой скоростью
            PWMA.start(Pw)
            PWMB.start(Pw)
```

```

# если есть ошибка,
    else:
# изменяем скважность ШИМ-сигналов каждого мотора:
# при er<0 левый крутим медленнее, правый быстрее,
# при er>0 левый крутим быстрее, правый медленнее
        PWMA.ChangeDutyCycle(Pw*(1+kP*der))
        PWMB.ChangeDutyCycle(Pw*(1-kP*der))
# отображаем все окна
# эти три строки можно удалить после отладки
    cv2.imshow('Frame', image)
    cv2.imshow('Crop', crop)
    cv2.imshow('only_object', only_object)
# очищаем RGB-массив для чтения следующего кадра
    raw.truncate(0)
# выходим, если нажать 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# робот - стоп
PWMA.stop()
PWMB.stop()
# освобождаем все ресурсы
camera.close()
GPIO.cleanup()
# закрываем все окна
cv2.destroyAllWindows()

```

Для отладки робота можно использовать любое поле с белым основанием и черной линией, задающей траекторию движения. Пару слов о настройке программы.

1. Первое, что надо сделать, — подобрать значение **Pw**, чтобы робот уверенно и без рывков начал движение. У нас это значение равно 25.
2. Установите держатель камеры так, чтобы в ее поле зрения не попадали посторонние объекты.
3. Если при запуске программы робот начал крутиться на месте, в строках:

```

PWMA.ChangeDutyCycle(Pw*(1+kP*der))
PWMB.ChangeDutyCycle(Pw*(1-kP*der))

```

знаки «+» и «-» поменяйте местами.

4. Подбирая коэффициент **kP**, добейтесь плавной реакции робота на выражах (в нашем случае это значение равно 0,5). Если робот «теряет» линию, увеличивайте коэффициент с шагом 0,05. Если он при движении дергается, уменьшайте значение **kP** с шагом 0,05.

При правильной настройке программы робот будет плавно двигаться вдоль черной линии. Результаты работы программы можно посмотреть здесь [19].

Итоги главы 3

В главе 3 мы познакомились с микрокомпьютером Raspberry Pi, с его аппаратными ресурсами и техническими характеристиками. Выяснили, что, несмотря на свою компактность, Raspberry Pi обладает вполне подходящей производительностью для большинства прикладных задач. Прошли «курс молодого бойца» по установке операционной системы Raspbian и библиотеки OpenCV на Raspberry. Разобрались с подключением модуля CSI-камеры для Raspberry и получили первые кадры видео с ее матрицы. Установили плату Raspberry Pi на робоплатформу AlphaBot. Научились управлять через интерфейс GPIO ходовыми моторами и программировать несложные элементы движения. И наконец, используя данные видеопотока с модуля CSI-камеры, создали программу движения робота вдоль черной линии.

Теперь мы знаем:

1. Что такое Raspberry Pi и на что он способен.
2. Что такое ОС Raspbian и порядок ее установки на Raspberry Pi.
3. Что такое терминал и как с ним работать.
4. Как установить библиотеку OpenCV на Raspberry Pi.
5. Как удаленно управлять Raspberry Pi.
6. Что такое модуль CSI-камеры для Raspberry и как его подключить.
7. Методы библиотеки **picamera** для захвата потока видеок кадров с модуля CSI-камеры.
8. Методы библиотеки **gpio** для управления пинами GPIO.
9. О существовании робоплатформы AlphaBot, конструкция которой хорошо подходит для установки платы Raspberry Pi.
10. Методы программной реализации ШИМ-сигнала для плавного управления ходовыми моторами AlphaBot.
11. Алгоритм «пропорциональный регулятор» для программирования движения робота AlphaBot вдоль черной линии по видеоданным модуля CSI-камеры.

Теперь мы умеем:

1. Подготовить Raspberry Pi к работе.
2. Установить на Raspberry Pi операционную систему Raspbian.
3. Запускать и выполнять некоторые команды из терминала.
4. Устанавливать на Raspbian библиотеку OpenCV-Python.
5. Подключать и настраивать работу модуля CSI-камеры для Raspberry.
6. Захватывать и отображать кадры из видеопотока модуля CSI-камеры с помощью методов библиотеки **picamera**.
7. Считывать и выводить сигналы на порты GPIO.
8. Настраивать GPIO для управления ходовыми моторами роботоплатформы AlphaBot.
9. Программировать движение роботоплатформы AlphaBot вдоль черной линии по видеоданным модуля CSI-камеры.

Глоссарий к главе 3

AlphaBot — робототехнический конструктор от Waveshare Electronics.

Bluetooth — открытый стандарт беспроводной связи с низким энергопотреблением для передачи данных между совместимыми устройствами.

cmake — утилита для автоматической сборки программы из исходного кода.

CSI (Camera Serial Interface) — компактный специализированный интерфейс для работы с цифровой камерой.

DSI (Display Serial Interface) — специализированный интерфейс для работы с жидкокристаллическими экранами.

Ethernet — технология пакетной передачи данных в компьютерных сетях.

GPIO (General-Purpose Input/Output) — интерфейс для связи между компонентами компьютерной системы и различными периферийными устройствами.

GPIO.cleanup() — метод, приводит все пины GPIO в исходное состояние.

GPIO.PWM() — метод, создает объект программной реализации сигнала ШИМ на указанном пине.

GPIO.setmode() — метод, устанавливает режим нумерации пинов GPIO.

GPIO.setup() — метод, устанавливает конфигурацию пинов GPIO.

HDMI (High Definition Multimedia Interface) — интерфейс передачи мультимедийных данных высокой четкости.

microSD — один из форм-факторов энергонезависимых ультракомпактных карт памяти (флеш-память) размером $11 \times 15 \times 1$ мм, весом около 1 г, относящихся к семейству карт памяти формата SD (Secure Digital).

PiCamera — пакет для работы с модулем CSI-камеры Raspberry.

Raspberry Pi — одноплатный компьютер компактного размера.

VNC (Virtual Network Computing) — платформонезависимая система удаленного доступа к рабочему столу компьютера, осуществляющая управление путем передачи нажатий клавиш на клавиатуре и движений мыши с одного компьютера на другой и ретрансляции содержимого экрана через компьютерную сеть.

Wi-Fi — технология беспроводной локальной сети для передачи цифровых потоков данных по радиоканалам с частотой 2,4–2,5 ГГц.

Терминал — окно для ввода текстовых команд в операционной системе Raspbian; системная утилита, обрабатывающая эти команды.

Список литературы к главе 3

1. Raspberry Pi // Официальный сайт разработчика Raspberry Pi. URL: <https://www.raspberrypi.org>
2. Raspberry Pi 3. Обзор и начало работы // Портал о микроконтроллере Raspberry Pi. URL: <https://myraspberrypi.ru/obzor-raspberrypi-3-odnoplattnik,-kotoryij-stal-polnoczennyim-kompyuterom.html>
3. Raspberry Pi 3. Обзор и начало работы. URL: <https://dmitrysnotes.ru/raspberrypi-3-obzor-i-nachalo-raboty>
4. Raspberry Pi. Downloads // Официальный сайт разработчика Raspberry Pi. URL: <https://www.raspberrypi.org/downloads/>
5. RPi SD-cards // URL: https://elinux.org/RPi_SD_cards
6. Подготовка карты памяти для Raspberry Pi // URL: <http://www.lexone.ru/raspberrypi/sdcard-preparation.html>
7. Подготовка SD-карты для Raspberry Pi // Мехатроника и робототехника. Республиканский портал мехатроники и робототехники. URL: <https://robo02.ru/2018/02/16/lesson-1-preparing-sd-card-for-raspberrypi/>
8. Raspberry Pi: установка Raspbian с помощью NOOBS // Robotclass. URL: <http://robotclass.ru/articles/raspberrypi-noobs-raspbian/>
9. Raspberry Pi. Downloads. NOOBS // Официальный сайт разработчика Raspberry Pi. URL: <https://www.raspberrypi.org/downloads/noobs/>

10. Open Source Computer Vision Library // GitHub. URL: <https://github.com/opencv/opencv>
11. Repository for OpenCV's extra modules // GitHub. URL: https://github.com/opencv/opencv_contrib
12. Installation in Linux // OpenCV. URL: https://docs.opencv.org/3.4.1/d7/d9f/tutorial_linux_install.html
13. Raspberry Pi 3: OpenCV + Python (#1) — установка библиотеки для использования с Python // CODIUS. URL: https://codius.ru/articles/Raspberry_Pi_3_OpenCV_часть_1
14. Установка OpenCV 3 на Raspberry Pi 3 // Robotclass. URL: <http://robotclass.ru/articles/raspberrypi-3-opencv-3-install>
15. Способы удаленного управления Raspberry Pi 3 // Портал о микроконтроллере Raspberry Pi. URL: <https://myraspberrypi.ru/sposobyi-udalennogo-upravleniya-raspberrypi-3.html>
16. Raspberry Pi 3 и SAMBA: удаленный доступ к файлам Ма- линки по локальной сети // CODIUS. URL: https://codius.ru/articles/Raspberry_Pi_3_и_SAMBA_удаленный_доступ_к_файлам_по_локальной_сети
17. VNC® Connect consists of VNC® Viewer and VNC® Server // Realvnc. URL: <https://www.realvnc.com/en/connect/download/viewer/windows/>
18. AlphaBot, Mobile robot development platform // Waveshare Electronics. URL: <https://www.waveshare.com/product/robotics/alphabot/platforms.htm>
19. Движение вдоль черной линии с помощью камеры Raspberry Pi // URL: <https://youtu.be/PhHK5OK1ENU>

Заключение

Современный IT-мир породил огромное множество прикладных задач, связанных с применением компьютерного зрения. Сейчас никого не удивляют цифровые фотокамеры, распознающие лица людей и определяющие пол и возраст, видеокамеры, регистрирующие нарушения правил дорожного движения, и т. д. В то же время область знаний, связанная с компьютерным зрением, постоянно развивается. Программная реализация инструментов для эффективного распознавания визуальной информации требует серьезных знаний в области математических методов и новых алгоритмов для быстрой обработки больших объемов данных. Программные коды оптимизируются под новые аппаратные платформы и операционные системы, появляются новые библиотеки и инструментальные средства. А это в свою очередь открывает горизонты для решения новых прикладных задач с помощью систем компьютерного зрения в самых различных областях.

Знание основ и принципов реализации компьютерного зрения позволяет сделать еще один шаг в развитии интеллектуальных роботизированных систем. Человек в своей жизни больше всего получает информацию визуально. Понимание принципов анализа и распознавания визуальной информации имеет огромное значение в робототехнике. Это совершенно иной уровень сенсорики и восприятия окружающего мира. И это совершенно иные функциональные возможности роботизированных систем и машин. Пройдет совсем немного времени, и машины с компьютерным зрением станут таким же обычным делом, как в свое время стали Интернет, компьютер или смартфон.

На страницах этой книги мы познакомились с языком программирования Python. В этом нам очень помог исполнитель Turtle. Безбрежное море существующих дополнительных пакетов и библиотек для Python помогло нам узнать, какой это мощный инструмент для программиста. И при этом процедура установки большинства пакетов

вполне доступна для тех, кто только начал изучать этот язык. А как много мы узнали благодаря библиотеке для компьютерного зрения OpenCV! Начав с простейших операций по открытию графических файлов и их трансформации, мы шаг за шагом продвигались вперед. Что такое цветовой фильтр, как с помощью него можно найти цветной объект на фото — это наши первые попытки понять, как работают алгоритмы компьютерного зрения. В процессе поиска цветных объектов мы открыли для себя, что цвета в памяти компьютера могут быть представлены по-разному. Узнали о существовании пространства цветов. Оказалось, что в пространстве HSV проще подобрать цветовой фильтр для обнаружения нужного объекта. И совсем по-другому стала восприниматься работа наших программ при обработке данных с видеопотока. С помощью программных инструментов контурного анализа мы научились выделять прямоугольные и эллиптические объекты, определять их размеры и отсеивать «шум». Наши программы превратились из «нескольких простеньких, в несколько строк» в хороший программный код. А потом мы познакомились с микрокомпьютером Raspberry Pi. Надо сказать, у «малышки-малинки» совсем даже неплохие характеристики с ее 4-ядерным процессором «на борту». Развернув операционную систему Raspbian и пакет OpenCV для Raspberry, мы подключили и поработали с модулем видеокамеры CSI. И в завершение запрограммировали свой первый робот на платформе AlphaBot, который двигался под управлением данных, получаемых из видеопотока CSI-камеры.

Надеемся, что материал книги будет полезен вам и поможет сделать первый шаг в изучении компьютерного зрения.

Минимальные системные требования определяются соответствующими требованиями программ Adobe Reader версии не ниже 11-й либо Adobe Digital Editions версии не ниже 4.5 для платформ Windows, Mac OS, Android и iOS; экран 10"

Электронное издание для дополнительного образования

Серия: «Школа юного инженера»

Шакирьянов Эдуард Данисович

**КОМПЬЮТЕРНОЕ ЗРЕНИЕ НА PYTHON®.
ПЕРВЫЕ ШАГИ**

Для детей среднего и старшего школьного возраста

Ведущий редактор *Т. Г. Хохлова*

Ведущий методист *А. А. Салахова*

Художественный редактор *В. А. Прокудин*

Технический редактор *Т. Ю. Федорова*

Корректор *И. Н. Панкова*

Компьютерная верстка: *В. И. Савельев*

Подписано к использованию 12.11.20.

Формат 155×225 мм

Издательство «Лаборатория знаний»

125167, Москва, проезд Аэропорта, д. 3

Телефон: (499) 157-5272

e-mail: info@pilotLZ.ru, <http://www.pilotLZ.ru>

КОМПЬЮТЕРНОЕ ЗРЕНИЕ НА PYTHON®

ПЕРВЫЕ ШАГИ



Шакирьянов Эдуард Данисович. Кандидат физико-математических наук, доцент кафедры цифровых технологий и моделирования Уфимского государственного нефтяного технического университета (УГНТУ), преподаватель Молодежного технопарка УГНТУ.

Еще совсем недавно создание устройств, оснащенных простейшим компьютерным зрением, было делом нешуточной сложности и требовало от разработчика больших интеллектуальных и материальных ресурсов. И вдруг все изменилось – на сцену вышли недорогие компактные микрокомпьютеры и общедоступное программное обеспечение. Теперь даже школьникам среднего и старшего возраста по плечу написать программный код для работы несложной системы с компьютерным зрением.

Книга, которую вы держите в руках, является учебным курсом для школьников, начинающих изучать компьютерное зрение с языком программирования Python® и библиотекой OpenCV. На ее страницах рассмотрены вопросы, раскрывающие особенности установки Python®, различных библиотек, в том числе OpenCV, и операционной системы Raspbian. Учебный материал изложен по отдельным темам:

- Программирование на языке Python®.
- Поиск и выделение цветных объектов на графическом изображении и в видеопотоке средствами OpenCV.
- Программирование колесной роботоплатформы под управлением Raspberry Pi® 3, оснащенной CSI-камерой.

Большую помощь читателю окажут многочисленные иллюстрации и листинги программных кодов, а также ссылки на источники и интернет-ресурсы.

Книга будет полезна школьникам среднего и старшего возраста, педагогам дополнительного образования и всем начинающим изучать компьютерное зрение с помощью языка программирования Python® и открытой библиотеки компьютерного зрения OpenCV-Python.



6+

