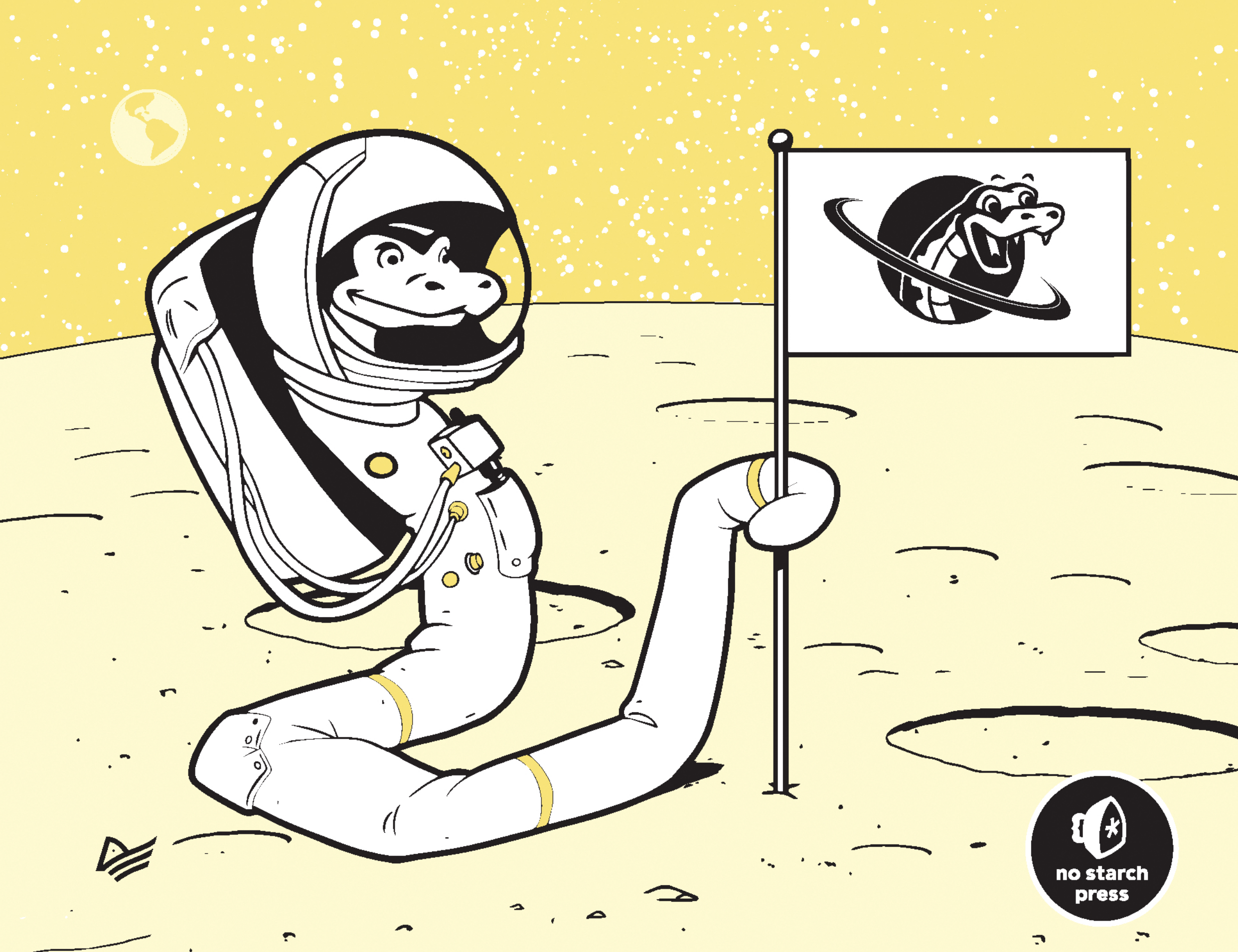
PYTHON ДЛЯ ХАКЕРОВ

НЕТРИВИАЛЬНЫЕ ЗАДАЧИ И ПРОЕКТЫ

ЛИ ВОГАН



REAL-WORLD PYTHON

A Hacker's Guide to Solving Problems with Code

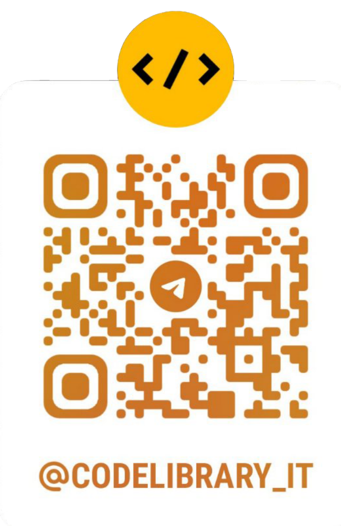
by Lee Vaughan

no starch press

San Francisco

PYTHON ДЛЯ ХАКЕРОВ

НЕТРИВИАЛЬНЫЕ ЗАДАЧИ И ПРОЕКТЫ



ЛИ ВОГАН

Санкт-Петербург • Москва • Минск

2023

Ли Воган

Python для хакеров. Нетривиальные задачи и проекты

Перевел с английского Д. Брайт

Научный редактор В. Кадочников

ББК 32.973.2-018.1
УДК 004.43

Воган Ли

В61 Python для хакеров. Нетривиальные задачи и проекты. — СПб.: Питер, 2023. — 384 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-2968-3

«Python для хакеров. Нетривиальные проекты и задачи» делает упор на реальные проекты, так что от экспериментирования с синтаксисом вы сразу перейдете к написанию полноценных программ. Развивая свои навыки разработки на Python, вы будете проводить научные опыты, изучать статистику и решать задачи, которые не давали покоя гениям на протяжении многих лет, и даже займетесь обнаружением далеких экзопланет.

Каждая глава начинается с четко поставленной цели и обсуждения способов решения задачи. Далее следует собственно миссия и стратегия действий, построенная таким образом, чтобы вы научились мыслить как программист. Вы будете руководить спасательной операцией береговой охраны, спланируете и осуществите полет космического корабля на Луну, реализуете ограничение доступа в секретную лабораторию с помощью распознавания лиц и не только это.

Программы, представленные в книге, не отпугнут даже новичков. Вы будете осваивать все более сложные техники и наращивать навыки написания кода. Справившись со всеми миссиями, вы будете готовы к самостоятельному решению любых сложных реальных задач с помощью Python.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ISBN 978-1718500624 англ. © 2021 by Lee Vaughan. Real-World Python: A Hacker's Guide to Solving Problems with Code, ISBN 9781718500624, published by No Starch Press Inc. 245 8th Street, San Francisco, California United States 94103.
Russian edition published under license by No Starch Press Inc.
ISBN 978-5-4461-2968-3 © Перевод на русский язык ООО «Прогресс книга», 2023
© Издание на русском языке, оформление ООО «Прогресс книга», 2023
© Серия «Библиотека программиста», 2023

Права на издание получены по соглашению с No Starch Press. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

Изготовлено в России. Изготовитель: ООО «Прогресс книга». Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург, Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.
Дата изготовления: 03.2023. Наименование: книжная продукция. Срок годности: не ограничен.
Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.
Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.
Подписано в печать 30.01.23. Формат 70×100/16. Бумага офсетная. Усл. п. л. 30,960. Тираж 1000. Заказ 0000.

Посвящается моему дяде, Кеннету П. Вогану.
Ты всегда и везде нес свет

Оглавление

Об авторе 13
О научных редакторах 14
Благодарности 15
От издательства 16

Введение 17
Для кого эта книга? 17
Почему Python? 18
План книги 18
Версия Python, платформа и IDE 20
Установка Python 21
Запуск Python 22
Использование виртуальной среды 24
Вперед! 24

Глава 1. Спасение моряков с помощью теоремы Байеса 25
Теорема Байеса 26
Проект #1. Поиск и спасение 29
Стратегия 31
Установка библиотек Python 31
Код для теоремы Байеса 34
Время сыграть 51
Итоги 53
Дополнительная литература 53
Усложняем проект. Более грамотный поиск 54
Усложняем проект. Поиск лучшей стратегии с помощью MCS 54
Усложняем проект. Вычисление вероятности обнаружения 54

Глава 2. Установление авторства с помощью стилометрии 57
Проект #2: «Собака Баскервилей», «Война миров» и «Затерянный мир» 58
Стратегия 58
Установка NLTK 60
Корпусы текстов 62
Код стилометрии 63
Итоги 81
Дополнительная литература 81
Практический проект: охота на собаку Баскервилей с помощью распределения 82
Практический проект: тепловая карта пунктуации 83
Усложняем проект: фиксирование частотности 84

Глава 3. Суммаризация текста с помощью обработки естественного языка 85
Проект #3. У меня есть мечта... суммаризация речи! 86
Стратегия 87
Веб-скрапинг 87
Код для «У меня есть мечта» 88
Проект #4. Суммаризация речи с помощью gensim 96
Установка gensim 97
Код для суммаризации речи «Заправляйте свою кровать» 97
Проект #5. Суммаризация речи с помощью облака слов 100
Модули Word Cloud и PIL 102
Код для создания облака слов 102
Итоги 109
Дополнительная литература 109
Усложняем проект: ночные игры 109
Усложняем проект: суммаризация суммаризаций 111
Усложняем проект: суммаризация повести 111
Усложняем проект: важно не только что ты говоришь, но и как! 113

Глава 4. Отправка суперсекретных сообщений с помощью книжного шифра 114
Одноразовый блокнот 115
Шифр «Ребекка» 117

Проект #6. Цифровой ключ к «Ребекке» 118
Стратегия 119
Код для шифрования 120
Отправка сообщений 130
Итоги 131
Дополнительная литература 131
Практический проект: составление графика символов 132
Практический проект: отправка секретов шифром времен Второй мировой войны 133

Глава 5. Поиск Плутона 135
Проект #7. Воссоздание блинк-компаратора 136
Стратегия 138
Данные 138
Код блинк-компаратора 139
Использование блинк-компаратора 153
Проект #8. Обнаружение астрономических транзиентов путем дифференцирования изображений 155
Стратегия 156
Код для детектора транзиентов 156
Использование детектора транзиентов 164
Итоги 164
Дополнительная литература 164
Практический проект: представление орбитальной траектории 164
Практический проект: найди отличия 165
Усложняем проект: сосчитаем звезды 166

Глава 6. Победа в лунной гонке с помощью «Аполлона-8» 167
Цель миссии «Аполлон-8» 168
Траектория свободного возврата 169
Задача трех тел 170
Проект #9. На Луну с «Аполлоном-8»! 171
Использование модуля turtle 171
Стратегия 175
Код программы для расчета свободного возврата «Аполлона-8» . . 176
Выполнение симуляции 192

Итоги ... 194
Дополнительная литература ... 194
Практический проект: симуляция шаблона поисков ... 195
Практический проект: запусти меня! ... 196
Практический проект: останови меня! ... 197
Усложняем проект: симуляция в истинном масштабе ... 198
Усложняем проект: реальный «Аполлон-8» ... 198

Глава 7. Выбор мест высадки на Марсе ... 199
Посадка на Марс ... 199
Карта MOLA ... 201
Проект #10. Выбор посадочных мест на Марсе ... 202
Стратегия ... 202
Код для выбора мест посадки ... 204
Результаты ... 222
Итоги ... 224
Дополнительная литература ... 224
Практический проект: убедимся, что рисунки становятся частью изображения ... 225
Практический проект: визуализация профиля высот ... 225
Практический проект: отображение в 3D ... 226
Практический проект: совмещение карт ... 227
Усложняем проект: три в одном ... 229
Усложняем проект: перенос прямоугольников ... 230

Глава 8. Обнаружение далеких экзопланет ... 231
Транзитная фотометрия ... 232
Проект #11. Симуляция транзита экзопланеты ... 234
Стратегия ... 235
Код для транзита ... 235
Эксперименты с транзитной фотометрией ... 242
Проект #12. Получение изображений экзопланет ... 245
Стратегия ... 245
Код для пикселизатора ... 247
Итоги ... 253
Дополнительная литература ... 253

Практический проект: обнаружение инопланетных мегаструктур 254
Практический проект: обнаружение транзита астероидов 256
Практический проект: добавление эффекта потемнения к краю 258
Практический проект: обнаружение пятен на звездах 260
Практический проект: обнаружение инопланетной армады 261
Практический проект: обнаружение планеты с Луной 262
Практический проект: измерение продолжительности экзопланетного дня 262
Усложняем проект: генерация динамической кривой блеска 263

Глава 9. Как различить своих и чужих 264
Обнаружение лиц на фотографиях 264
Проект #13. Программирование робота-часового 266
Стратегия 269
Код 269
Результаты 282
Обнаружение лиц в видеопотоке 283
Итоги 287
Дополнительная литература 287
Практический проект: размытие лиц 288
Усложняем проект: обнаружение кошачьих мордочек 289

Глава 10. Ограничение доступа по принципу распознавания лиц 290
Распознавание лиц с помощью LBPH 290
Схема распознавания лиц 291
Извлечение гистограмм локальных бинарных шаблонов 293
Проект #14. Ограничение доступа к инопланетному артефакту 296
Стратегия 296
Поддержка модулей и файлов 296
Код для захвата видео 297
Код для обучения алгоритма распознавания лиц 302
Код для прогнозирования лиц 305
Результаты 308
Итоги 309
Дополнительная литература 309
Усложняем проект: добавление пароля и видеозахвата 310

Усложняем проект: похожие лица и близнецы 311
Усложняем проект: машина времени 311

Глава 11. Создание интерактивной карты побега от зомби 312
Проект #15. Визуализация плотности населения с помощью хороплетной карты 312
Стратегия 314
Библиотека анализа данных 315
Библиотеки bokeh и holoviews 317
Установка pandas, bokeh и holoviews 318
Работа с данными по уровню безработицы и плотности населения в округах и штатах 318
Разбираем код holoviews 320
Код для отрисовки хороплетной карты 323
Планирование маршрута 332
Итоги 336
Дополнительная литература 336
Усложняем проект: отображение на карте изменения численности населения США 337

Глава 12. Находимся ли мы в компьютерной симуляции? 338
Проект #16. Жизнь, Вселенная и пруд черепахи Йертл 339
Код симуляции пруда 339
Следствия симуляции пруда 342
Измерение затрат на пересечение строк или столбцов сетки 344
Результаты 347
Стратегия 348
Итоги 349
Дополнительная литература 349
Дополнение 350
Усложняем проект: поиск безопасного места в космосе 350
Усложняем проект: а вот и Солнце 351
Усложняем проект: взгляд глазами собаки 351
Усложняем проект: кастомизированный поиск слов 351
Усложняем проект: оптимизация праздничного показа слайдов 352
Усложняем проект: что за сложную паутину мы плетем 352
Усложняем проект: идем вещать с горы 352

Решения для практических проектов 353
Глава 2. Определение авторства с помощью стилометрии 353
Охота на собаку Баскервилей с помощью распределения 353
Тепловая карта пунктуации 354
Глава 4. Отправка суперсекретных сообщений с помощью книжного шифра 355
Составление графика символов 355
Отправка секретов шифром времен Второй мировой войны 356
Глава 5. Поиск Плутона 359
Представление орбитальной траектории 359
В чем разница? 360
Глава 6. Победа в лунной гонке с помощью «Аполлона-8» 362
Симуляция шаблона поисков 362
Заведи меня! 363
Останови меня! 366
Глава 7. Выбор мест высадки на Марсе 368
Убеждаемся, что рисунки становятся частью изображения 368
Визуализация профиля высоты 368
Отображение в 3D 369
Совмещение карт 370
Глава 8. Обнаружение далеких экзопланет 374
Обнаружение инопланетных мегаструктур 374
Обнаружение транзита астероидов 375
Добавление эффекта потемнения к краю 377
Обнаружение инопланетной армады 378
Обнаружение планеты с Луной 379
Измерение продолжительности экзопланетного дня 382
Глава 9. Как различить своих и чужих 383
Размытие лиц 383
Глава 10. Ограничение доступа по принципу распознавания лиц 383
Усложняем проект: добавление пароля и видеозахвата 383

Об авторе

Ли Воган — программист, поклонник поп-культуры, консультант, автор книги «Impractical Python Projects»[1] (No Starch Press, 2018). За десятилетия работы научным руководителем в компании ExxonMobil он занимался проектированием и анализом компьютерных моделей, разрабатывал и тестировал программное обеспечение и, кроме того, обучал геофизиков и инженеров.

Обе свои книги, «Непрактичный Python» и «Python для хакеров», он написал для тех, кто самостоятельно изучает Python и хочет отточить свое мастерство, выполняя увлекательные и нетривиальные проекты.

---

[1] «Непрактичный Python. Занимательные проекты для тех, кто хочет поумнеть».

О научных редакторах

Крис Крен (Chris Kren) окончил Университет Южной Алабамы со степенью магистра в сфере информационных систем. Сейчас он занимается кибербезопасностью и часто использует Python для отчетов, анализа данных и автоматизации.

Эрик Тодд Мортенсон (Eric Todd Mortenson) получил докторскую степень по математике в Висконсинском университете в Мадисоне. Он занимался исследованиями и преподаванием в Университете штата Пенсильвания, Университете Квинсленда, а также в Математическом институте Макса Планка. Сейчас он работает доцентом на факультете математики и компьютерных наук в Санкт-Петербургском государственном университете.

Благодарности

Команда No Starch Press работала над книгой во время пандемии — и совершила очередной трудовой подвиг. Их можно смело назвать профессионалами, которым нет равных, и эта книга появилась на свет лишь благодаря их усилиям. Выражаю всем сотрудникам свою глубочайшую признательность и уважение.

Также благодарю Криса Крена и Эрика Эвенчика (Eric Evenchick) за их ревью кода, Джозефа Б. Пола (Joseph B. Paul), Сару и Лору Воган — за их энтузиазм в косплее, а также Ханну Воган — за полезные фотографии.

Отдельное спасибо Эрику Т. Мортенсону за его подробную научную рецензию, содержащую множество полезных идей и дополнений. Эрик предложил добавить главу, посвященную байесовскому правилу, и предоставил множество практических проектов, а также дополнительных задач, включая применение к байесовским моделям методов Монте-Карло, подведение итогов романа по главам, моделирование взаимосвязей между Луной и «Аполлоном-8», просмотр Марса в 3D, вычисление кривой блеска для экзопланеты, обладающей луной, и некоторые другие. Благодаря его усилиям эта книга стала намного лучше.

В завершение выражаю благодарность участникам с ресурса stackoverflow.com. Одна из лучших особенностей Python заключается в его обширном и многогранном сообществе пользователей. Неважно, какой вопрос у вас возникнет, кто-нибудь на него обязательно ответит. Неважно, насколько странную задачу вы решаете, кто-нибудь наверняка уже делал что-то подобное. И всех этих людей вы можете найти на Stack Overflow.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Введение

Если вы освоили основы работы с Python, то уже готовы писать полноценные программы для решения настоящих задач. В книге «Python для хакеров. Нетривиальные задачи и проекты» вы напишете такие программы, чтобы победить в лунной гонке на «Аполлоне-8», помочь Клайду Томбо открыть Плутон, выбрать посадочные места на Марсе, обнаружить экзопланеты, отправить суперсекретные сообщения друзьям, сразиться с ужасными мутантами, спасти моряков после кораблекрушения, убежать от зомби и сделать еще много другого — и все это с помощью языка программирования Python. Вы будете применять мощные техники компьютерного зрения, обработку естественного языка и научные модули, такие как OpenCV, NLTK, `NumPy`, `pandas`, `matplotlib`, а также многие другие библиотеки, созданные для облегчения жизни программистов.

Для кого эта книга?

Можно рассматривать эту книгу как пособие по Python для второкурсников. Это не руководство по основам языка, а возможность продолжить обучение, работая над реальными проектами. Таким образом, вам не придется тратить деньги и место на полке, только чтобы освежить в памяти уже известные принципы. Но я все равно буду объяснять каждый этап проекта, давать подробные инструкции по использованию библиотек и модулей, включая их установку.

Эти проекты заинтересуют всех, кто хочет использовать программирование для экспериментов, проверки теорий, моделирования природных явлений или

просто для развлечения. По мере выполнения проектов вы будете накапливать знания о библиотеках Python и модулях, а также узнаете новые полезные приемы, функции и техники. Мы не будем зацикливаться на отдельных фрагментах кода; вместо этого вы научитесь создавать полноценные программы для решения реальных задач, используя реальные данные.

Почему Python?

Python — это высокоуровневый интерпретируемый язык общего назначения. Он свободно распространяемый, интерактивный и совместимый со всеми ведущими платформами, а также микроконтроллерами, например с Raspberry Pi. Python поддерживает и функциональное, и объектно-ориентированное программирование, а также способен взаимодействовать с кодом, написанным на других языках, например на C++.

Поскольку Python вполне доступен для начинающих и полезен для экспертов, он широко применяется в школах, университетах, крупных корпорациях, финансовых учреждениях и практически во всех областях науки. Сегодня этот язык наиболее популярен для машинного обучения, в областях, связанных с наукой о данных и искусственным интеллектом.

План книги

Итак, краткий обзор глав этой книги. Вам не обязательно изучать их все последовательно, но я буду объяснять новые модули и техники более подробно при их первом упоминании.

- **Глава 1. Спасение моряков с помощью теоремы Байеса.** Используем теорему Байеса, чтобы эффективно направить береговую охрану для поиска и спасения моряков у мыса Python. Набираемся опыта в применении OpenCV, `NumPy` и модуля `itertools`.
- **Глава 2. Установление авторства с помощью стилометрии.** Используем обработку естественного языка для определения автора романа «Затерянный мир» — был ли это сэр Артур Конан Дойл или же Герберт Джордж Уэллс? Практикуемся в работе с NLTK, `matplotlib` и такими стилометрическими техниками, как стоп-слова, части речи, лексическое богатство и коэффициент Жаккара.
- **Глава 3. Суммаризация текста с помощью обработки естественного языка.** Делаем скрапинг известных речей из интернета и автоматическое обобщение их важных моментов. Преобразуем текст романа в аннотацию для рекламы или промоматериала. Расширяем навыки

работы с `BeautifulSoup`, `Requests`, `regex`, NLTK, `Collections`, `wordcloud` и `matplotlib`.

- **Глава 4. Отправка суперсекретных сообщений с помощью книжного шифра.** Делимся невзламываемыми шифрами с друзьями путем цифрового воссоздания метода «одноразового блокнота», использованного в шпионском романе Кена Фоллетта «Ключ к Ребекке». Учимся работать с модулем `Collections`.
- **Глава 5. Поиск Плутона.** Воссоздаем блинк-компаратор, с помощью которого Клайд Томбо открыл Плутон в 1930 году. Затем используем современное компьютерное зрение для автоматического поиска и отслеживания слаборазличимых транзиентов, таких как кометы и астероиды, перемещающихся относительно звездного поля. Получаем опыт работы с OpenCV и `NumPy`.
- **Глава 6. Победа в лунной гонке с помощью «Аполлона-8».** Принимаем участие в приключении и помогаем США победить в лунной гонке, первыми достигнув Луны на корабле «Аполлон-8». Составляем и реализуем грамотный план обратного полета, который в ретроспективе убедил NASA отправиться в полет на год раньше и по факту нанес удар по советской космической программе. Набираемся опыта в использовании модуля `turtle`.
- **Глава 7. Выбор мест высадки на Марсе.** Оцениваем потенциальные места посадки для марсохода на основе реальных задач миссии. Отображаем предполагаемые точки на карте Марса вместе с их сводной статистикой. Совершенствуем навыки работы с OpenCV, Python Imaging Library, `NumPy` и `tkinter`.
- **Глава 8. Обнаружение далеких экзопланет.** Моделируем проход экзопланеты на фоне ее солнца, отображаем график итоговых изменений относительной яркости и оцениваем диаметр этой планеты. В завершение симулируем прямое наблюдение экзопланеты с помощью нового телескопа Джеймса Уэбба, включая оценку длительности ее дня. Используем OpenCV, `NumPy` и `matplotlib`.
- **Глава 9. Как различить своих и чужих.** Программируем роботизированную пушку-стража на визуальное распознавание космических пехотинцев и злых мутантов. Применяем OpenCV, `NumPy`, `playsound`, `pyttsx` и `datetime`.
- **Глава 10. Ограничение доступа по принципу распознавания лиц.** Реализуем ограничение доступа в секретную лабораторию через распознавание лиц. Используем OpenCV, `NumPy`, `playsound`, `pyttsx` и `datetime`.
- **Глава 11. Создание интерактивной карты побега от зомби.** Создаем карту плотности популяции, которая поможет выжившим в ТВ-шоу «Ходячие

мертвецы» выбраться из Атланты на безопасную территорию Запада США. Совершенствуемся в работе с `pandas`, `bokeh`, `holoviews` и `webbrowser`.

- **Глава 12. Находимся ли мы в компьютерной симуляции?** Определяем способ для симулированных существ — возможно, это мы сами — отыскать свидетельства того, что они живут в компьютерной симуляции. Используем для этого `turtle`, `statistics` и `perf_counter`.

Каждая глава завершается как минимум одним практическим или усложненным проектом. Их решения вы найдете в приложении или онлайн. Учтите, что эти решения не единственные и не обязательно лучшие. Так что, возможно, вам удастся придумать что-то более эффективное.

Что же касается усложненных проектов, то здесь все зависит только от вас. В них я реализую принцип «плыви или тони», который здорово помогает в обучении. Надеюсь, моя книга сможет мотивировать вас на создание собственных проектов, а такие задачи играют роль триггеров, которые взбудоражат ваше воображение.

Можете скачать весь код книги, включая решения к практическим проектам, с сайта https://nostarch.com/real-world-python/. Там же я публикую список опечаток и всевозможные будущие обновления.

Невозможно написать подобную книгу без недочетов. Если вы обнаружите, что в книге что-то не так, пожалуйста, отправьте описание проблемы издателю по адресу errata@nostarch.com. Мы будем вносить все необходимые правки в список опечаток и включим исправление в переиздания, а вы получите вечное признание и славу.

Версия Python, платформа и IDE

Все проекты этой книги я создавал на Python v3.7.2 в Microsoft Windows 10. Если вы используете другую операционную систему, то это не проблема: там, где необходимо, я предлагаю совместимые модули для других платформ.

Примеры кода в книге взяты либо из текстового редактора Python IDLE, либо из интерактивной оболочки. IDLE (Integrated Development and Learning Environment) означает *«интегрированная среда разработки и обучения»*. Это та же *интегрированная среда разработки* (IDE), но с добавленной *L*, которая делает акроним созвучным фамилии актера Eric Idle, участника творческой группы *Monty Python*. Интерактивная оболочка, также называемая *интерпретатором*, — окно, позволяющее вам мгновенно выполнять команды и тестировать код, не создавая файл.

У IDLE множество недостатков, например отсутствует нумерация строк, но при этом она бесплатна и связана с Python, что дает каждому доступ к ней. Вы можете без проблем использовать любую IDE по своему желанию. Среди наиболее

популярных могу назвать Visual Studio Code, Atom, Geany (произносится «джини»), PyCharm и Sublime Text. Они работают в разных операционных системах, включая Linux, macOS и Windows. Еще одна IDE, PyScripter, работает только в Windows. Подробный список доступных редакторов Python и совместимых платформ вы найдете на https://wiki.python.org/moin/PythonEditors/.

Установка Python

Вы можете установить Python на свою машину через дистрибутив. Если же вы решите это сделать напрямую, то инструкции для вашей операционной системы вы найдете на https://www.python.org/downloads/. На машинах с Linux и macOS Python уже обычно предустановлен. С каждой новой версией языка некоторые возможности в него добавляются, а другие исключаются, поэтому я рекомендую обновить вашу версию, если она ниже v3.6.

Щелчок на кнопке скачивания на сайте Python (рис. 1) по умолчанию устанавливает 32-битный Python.

Рис. 1. Страница скачивания Python.org с удобной кнопкой для платформы Windows

Если же вам нужна 64-битная версия, то промотайте страницу вниз до списка конкретных версий (рис. 2) и щелкните на ссылке с тем же номером версии.

Откроется окно, показанное на рис. 3. Здесь щелкните по 64-битному исполняемому файлу, который запустит мастер установки. Следуйте инструкциям и соглашайтесь с настройками по умолчанию.

Для реализации некоторых проектов в этой книге требуются нестандартные библиотеки, которые придется устанавливать отдельно. Это несложно, но можно все упростить, установив дистрибутив Python, который эффективно загружает

и управляет сотнями библиотек Python. Это как все покупки сделать в одном магазине. Менеджеры пакетов в таких дистрибутивах будут автоматически находить и скачивать последние версии, включая все необходимые зависимости.

Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.7.7	March 10, 2020	Download	Release Notes
Python 3.8.2	Feb. 24, 2020	Download	Release Notes
Python 3.8.1	Dec. 18, 2019	Download	Release Notes
Python 3.7.6	Dec. 18, 2019	Download	Release Notes
Python 3.6.10	Dec. 18, 2019	Download	Release Notes
Python 3.5.9	Nov. 2, 2019	Download	Release Notes
Python 3.5.8	Oct. 29, 2019	Download	Release Notes

Рис. 2. Список версий на странице скачивания Python.org

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		2ee10f25e3d1b14215d56c3882486fcf	22973527	SIG
XZ compressed source tarball	Source release		93df27aec0cd18d6d42173e601ffbbfd	17108364	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	5a95572715e0d600de28d6232c656954	34479513	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	4ca0e30f48be690bfe80111daee9509a	27839889	SIG
Windows help file	Windows		7740b11d249bca16364f4a45b40c5676	8090273	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	854ac011983b4c799379a3baa3a040ec	7018568	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a2b79563476e9aa47f11899a53349383	26190920	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	047d19d2569c963b8253a9b2e52395ef	1362888	SIG
Windows x86 embeddable zip file	Windows		70df01e7b0c1b7042aabb5a3c1e2fbd5	6526486	SIG
Windows x86 executable installer	Windows		ebf1644cdc1eeeebacc92afa949cfc01	25424128	SIG
Windows x86 web-based installer	Windows		d3944e218a45d982f0abcd93b151273a	1324632	SIG

Рис. 3. Список файлов для Python 3.8.2 на Python.org

К популярным дистрибутивам относится Anaconda от Continuum Analytics. Можете скачать его с https://www.anaconda.com/. Еще один интересный дистрибутив — Enthought Canopy, хотя бесплатной является лишь его базовая версия. Независимо от того, установите вы библиотеки Python отдельно или через дистрибутив, с проработкой проектов из книги не должно возникнуть проблем.

Запуск Python

После установки Python должен отобразиться в списке приложений операционной системы. При его запуске появится окно оболочки (показано на заднем

плане рис. 4). Можете использовать эту интерактивную среду для запуска и тестирования сниппетов кода. Однако для написания более крупных программ вы будете использовать текстовый редактор, который позволяет сохранять код (рис. 4, на переднем плане).

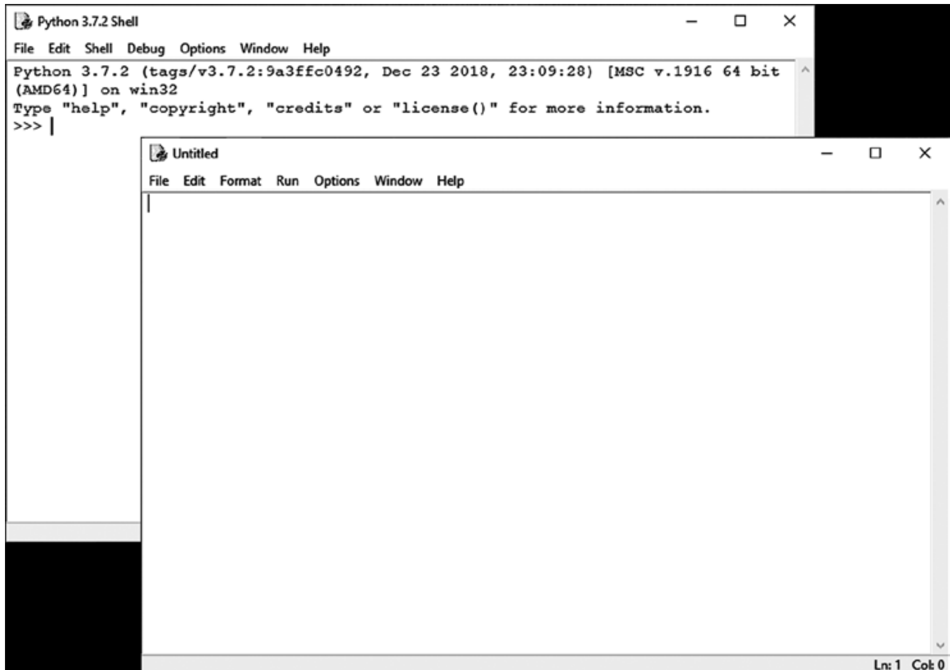


Рис. 4. Окно оболочки Python (задний план) и текстовый редактор (передний план)

Для создания нового файла в редакторе IDLE щелкните File ▸ New File. Чтобы открыть существующий файл, щелкните File ▸ Open или File ▸ Recent Files. Отсюда можно выполнять код через Run ▸ Run Module или нажатием F5 после выбора окна редактора. Обратите внимание, что ваша среда может выглядеть не так, как на рис. 4, если вы выбрали пакетный менеджер вроде Anaconda или IDE вроде PyCharm.

Программу Python также можно запускать, указав ее имя в PowerShell или Terminal. Для этого потребуется перейти в каталог, где эта программа расположена. Например, если вы не запустили Windows PowerShell из правильного каталога, то вам потребуется изменить путь каталога при помощи команды `cd` (рис. 5).

Рис. 5. Изменение каталогов и запуск программы Python в Windows PowerShell

Подробности можно узнать на https://pythonbasics.org/execute-python-scripts/.

Использование виртуальной среды

Ну и последнее. Вы можете устанавливать зависимости для каждой главы в отдельной виртуальной среде. В Python *виртуальная среда* — это самостоятельное дерево каталогов, включающее установку Python и ряд дополнительных пакетов. Такие среды полезны при работе с несколькими версиями Python, так как некоторые библиотеки могут быть совместимы с одной версией, но не с другими. Кроме того, это дает возможность работать с проектами, требующими разные версии одной библиотеки. Раздельное их хранение избавляет от проблем совместимости.

Проекты из книги не требуют использования виртуальных сред, и если вы будете следовать моим инструкциям, то установите необходимые библиотеки для всей системы. Однако если вам действительно нужно изолировать пакеты от операционной системы, то рассмотрите возможность установки отдельной виртуальной среды для каждой главы книги (подробнее — на https://docs.python.org/3.8/library/venv.html#module-venv и https://docs.python.org/3/tutorial/venv.html).

Вперед!

Многие проекты для этой книги созданы на основе статистических и научных принципов, которые известны уже сотню лет, но плохо реализуются вручную. Но с появлением персональных компьютеров в 1975 году наши возможности хранить информацию, обрабатывать ее и делиться ею возросли на много порядков.

За 200 000 лет истории человечества только живущим в последние 45 лет дана привилегия использовать эти магические устройства и реализовывать немыслимые когда-то идеи.

Так воспользуемся же этой возможностью по максимуму! На следующих страницах вы с легкостью решите задачи, которые не давали покоя гениям прошлого. Вы прикоснетесь к некоторым из удивительнейших возможностей, которые появились лишь недавно. И возможно, вы даже начнете предвидеть грядущие открытия.

1

Спасение моряков с помощью теоремы Байеса

Где-то в 1740 году британский пресвитерианский священник Томас Байес решил математически доказать существование Бога. Его гениальное решение, ныне известное как *теорема Байеса* (она же формула Байеса), сейчас считается одной из наиболее успешных статистических концепций всех времен. Однако на протяжении 200 лет ее по большей части игнорировали, потому что сложные математические вычисления было очень трудно производить вручную. Для того чтобы в полной мере оценить потенциал теоремы Байеса, потребовалось изобрести компьютер. Теперь благодаря нашим быстрым процессорам она стала ключевым элементом в науке о данных и в сфере машинного обучения.

Поскольку правило Байеса дает математически верный способ учитывать новые данные и пересчитывать оценку вероятности события, оно применяется практически во всех областях деятельности человека, начиная со взлома кодов и прогноза, кто победит на президентских выборах, и заканчивая расчетом того, как увеличится число инфарктов при повышении холестерина. Перечень областей, в которых применяется теорема Байеса, с легкостью займет целую главу книги. Но так как для нас важнее всего спасение жизни людей, мы рассмотрим, как использовать эту теорему для помощи потерявшимся морякам.

В этой главе мы создадим игру-симуляцию поисково-спасательной операции. Игроки, действующие от лица спасателей береговой охраны, будут использовать

теорему Байеса, чтобы принять решения, которые позволят как можно быстрее обнаружить моряков. Мы начнем с известных инструментов из области data science и компьютерного зрения — библиотеки Open Source Computer Vision Library (OpenCV) и `NumPy`.

Теорема Байеса

Теорема Байеса помогает исследователям определить вероятность какого-либо события при условии, что произошло другое взаимосвязанное с ним событие. Как сказал великий французский математик Лаплас: «Вероятность причины с учетом события пропорциональна вероятности события с учетом его причины». Базовая формула такова:

$$P(A/B) = \frac{P(B/A)P(A)}{P(B)}.$$

Здесь A — это гипотеза, а B — это данные. $P(A/B)$ означает вероятность A с учетом B. Предположим, нам известно, что конкретный тест на некий вид рака не всегда точен и может давать ложноположительные результаты, которые указывают, что рак у пациента есть, хотя на самом деле его нет.

Выражение Байеса в таком случае выглядит так:

$$\left(\begin{array}{l}\text{Вероятность рака}\\ \text{при положительном}\\ \text{тесте}\end{array}\right) = \left(\begin{array}{l}\text{Вероятность}\\ \text{положительного}\\ \text{теста у больных}\\ \text{раком}\end{array}\right) \times \left(\frac{\text{Вероятность наличия рака}}{\text{Вероятность положительного теста}}\right).$$

Начальная вероятность будет основана на клинических исследованиях. Например, 800 из 1000 человек, болеющих раком, могут получить положительный результат, а 100 из 1000 — ошибочный. На основе показателей заболеваемости общая вероятность наличия у конкретного человека рака может составлять всего 50 из 10 000. Итак, если общая вероятность наличия рака мала, а общая вероятность получения положительного теста относительно высока, то вероятность наличия рака при положительном тесте снижается. Если в исследованиях зафиксирована частота ошибочных результатов тестирования, то правило Байеса может скорректировать оценку измерений.

Теперь, когда принцип применения теоремы ясен, взгляните на рис. 1.1, где показаны различные варианты, следующие из теоремы Байеса, на примере заболевания раком.

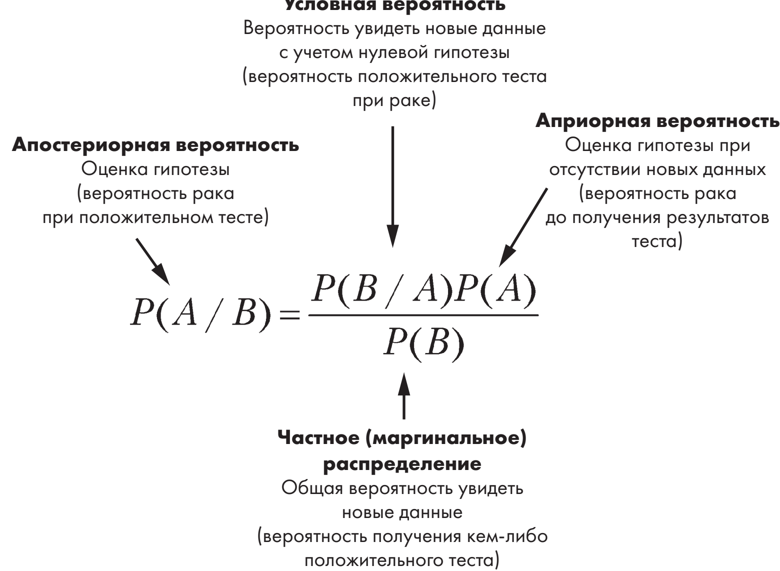


Рис. 1.1. Теорема Байеса: определение терминов на примере заболевания раком

Рассмотрим следующий пример: женщина потеряла где-то дома свои очки для чтения. Она помнит, что последний раз была в них, когда находилась в рабочем кабинете, поэтому первым делом идет искать в кабинет. Очки ей найти не удается, но она видит чайную чашку и вспоминает, что ходила на кухню. В этот момент ей надо сделать выбор: более тщательно обыскать кабинет или же пойти поискать на кухню. Она решает отправиться на кухню. Таким образом, она, сама не зная того, приняла байесовское решение.

В кабинет она отправилась, так как считала, что вероятность обнаружения очков там наиболее высока. На языке Байеса эта начальная вероятность обнаружения очков в кабинете называется *априорной*. После беглого осмотра кабинета она изменила свое решение на основе двух новых элементов информации: ей не удалось сразу обнаружить очки, но она увидела чашку. Этот фактор называется *байесовским выводом* — когда по мере появления дополнительных фактов вычисляется новая апостериорная оценка ($P(A/B)$ на рис. 1.1).

Представим, что женщина решила использовать при поиске теорему Байеса. Она определила вероятность нахождения очков в кабинете или на кухне и эффективность своих поисков в этих двух помещениях. Теперь вместо интуитивных ощущений ее решения опираются на математическую модель, которую можно постепенно обновлять, если дальнейшие поиски не дадут результата.

На рис. 1.2 показаны вероятности поиска очков.

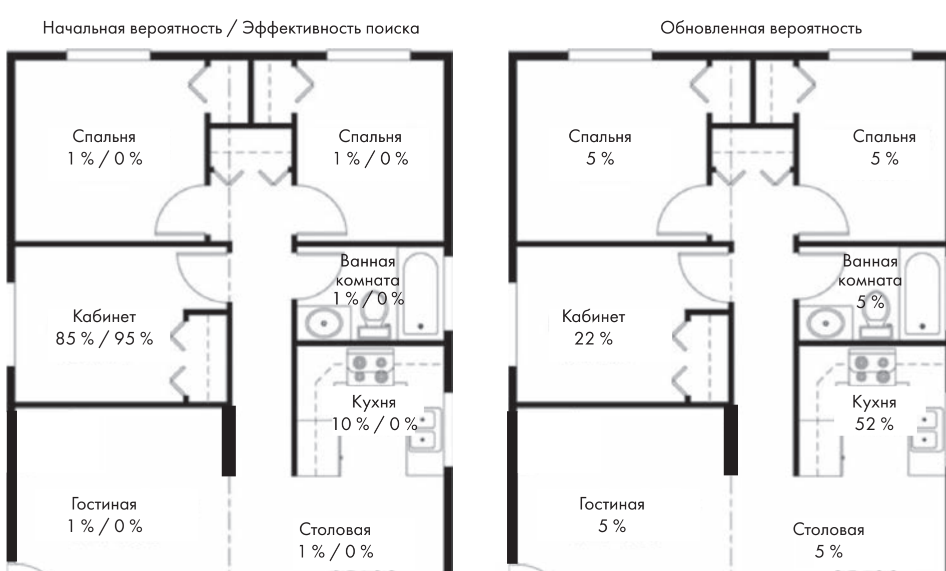


Рис. 1.2. Начальные вероятности того, что очки находятся в каждом помещении, и эффективности поиска (слева) в сравнении с обновленными целевыми вероятностями (справа)

Левая схема показывает начальную ситуацию. Правая — ситуацию после применения правила Байеса. Предположим, что изначально шанс найти очки в кабинете составлял 85 %, а на кухне — 10 %. Другим комнатам присвоена вероятность 1 %, потому что правило Байеса не может обновлять целевые вероятности, равные нулю (к тому же всегда есть небольшой шанс, что женщина оставила очки в одной из этих комнат).

На левой схеме числа после слэша показывают *вероятность эффективности поиска (search effectiveness probability, SEP)*, то есть оценку эффективности осмотра области. Так как женщина обыскала пока только кабинет, это значение для всех остальных комнат равно нулю. После байесовского вывода (обнаружения чашки) женщина может повторно вычислить вероятности на основе результатов поиска (показано справа). Теперь кухня становится наиболее предпочтительным местом для поиска, но при этом также возрастает вероятность и для других комнат.

Человеческая интуиция подсказывает: если что-то лежит не там, где мы рассчитываем, то шансы на то, что оно находится где-то в другом месте, возрастают. Теорема Байеса это учитывает, в связи с чем повышается вероятность того, что очки находятся в какой-то из других комнат. Однако это может произойти, только если изначально была вероятность их нахождения в другой комнате.

Формула для вычисления вероятности того, что очки находятся в заданной комнате, с учетом эффективности поиска выглядит так:

$$P(G / E) = \frac{P(E/G)P_{\text{prior}}(G)}{\Sigma P(E/G9)P_{\text{prior}}(G9)}.$$

Здесь G — это вероятность того, что очки находятся в комнате, E —эффективность поиска, а P_{prior} — априорная, или начальная, оценка вероятности до получения новых данных.

Вы можете получить обновленную вероятность того, что очки находятся в кабинете, подставив целевую вероятность и вероятность поиска в уравнение таким образом:

$$\frac{0{,}85\times(1-0{,}95)}{(0{,}85\times(1-0{,}95)+0{,}1\times(1-0)+0{,}01\times(1-0)+0{,}01\times(1-0)+0{,}01\times(1-0)+0{,}01\times(1-0)+0{,}01\times(1-0)}.$$

Как видите, простая математика может быстро стать неподъемной, если делать вычисления вручную. К счастью, мы живем в чудесную компьютерную эпоху, что позволяет поручить всю нудную работу Python.

Проект #1. Поиск и спасение

Сейчас вы напишете на Python программу, которая использует теорему Байеса для поиска одинокого моряка, пропавшего у мыса Python. Как руководитель поисково-спасательных операций береговой охраны вы уже опросили его жену и определили, когда и где моряка засекли в последний раз — после этого уже прошло шесть часов. Он сообщил по радио, что покидает тонущий корабль, но нет никаких данных о том, находится ли он в спасательной лодке или дрейфует в море. Воды у мыса теплые, но если он находится в воде, то примерно через 12 часов наступит переохлаждение организма. При условии, что он одет в спасательный жилет и ему благоволит судьба, у него есть шансы продержаться три дня.

У мыса Python сложные океанические течения (рис. 1.3), сейчас дует юго-западный ветер. Видимость хорошая, но волнение почти не позволяет разглядеть человеческую голову.

Рис. 1.3. Океанические течения у мыса Python

В реальной жизни вы передали бы всю информацию в систему оптимального планирования поисково-спасательных операций (Search and Rescue Optimal Planning System, SAROPS) береговой охраны. Это программное обеспечение учитывает такие факторы, как ветер, приливы и течения, находится ли тело в воде или на лодке и пр. На основе этих данных программа генерирует прямоугольные области поиска, вычисляет начальные вероятности нахождения моряка в каждой из них и составляет график наиболее эффективных траекторий полетов.

Мы предположим, что SAROPS определила три области поиска. Все, что нужно сделать, — это написать программу, которая применяет правило Байеса. При этом ресурсов у вас хватает на поиск в двух из трех областей за день. Вам нужно решить, как эти ресурсы распределить. Это очень трудно, но у вас есть мощный помощник — теорема Байеса.

ЗАДАЧА

Создать поисково-спасательную игру, в которой теорема Байеса используется для определения следующего шага при выполнении поиска.

Стратегия

Поиск моряка подобен поиску очков в предыдущем примере. Вы начнете с базовых целевых вероятностей его местоположения и будете обновлять их по мере получения результатов поиска. Если вы достигнете высокой эффективности поиска в области, но ничего не найдете, то вероятность того, что моряк находится в другой области, увеличится.

Здесь, как и в реальной жизни, есть два варианта, когда все может пойти не так: тщательный поиск в области, не увенчавшийся успехом, или плохой поиск, при котором усилия целого дня затрачиваются впустую. Если приравнять это к показателям эффективности поиска, то в первом случае вы можете получить SEP, равный 0.85, но не произвести поиск моряка в оставшихся 15 % области. Во втором же случае SEP будет 0.2 и у вас останутся неисследованными 80 % области.

Можете представить себе дилемму, которая возникает у руководителей реальных спасательных операций? Послушаете ли вы собственную интуицию, проигнорировав Байеса? Будете ли придерживаться чистой холодной логики Байеса, так как верите, что это наилучший вариант? А может, будете действовать целесообразно и подстрахуете свою карьеру и репутацию, строго следуя правилу Байеса, даже когда в нем не уверены?

В помощь игроку вы, используя библиотеку OpenCV, создадите интерфейс для работы с программой. Несмотря на то что интерфейс может быть простым, например в виде встроенного в оболочку меню, вам также понадобится карта мыса и областей поиска. На карте вы будете отмечать последнее известное местоположение моряка, а также место его обнаружения. Библиотека OpenCV является отличным ресурсом для этой игры, потому что позволяет использовать изображения, а также добавлять текст и рисунки.

Установка библиотек Python

OpenCV — это крупнейшая в мире библиотека компьютерного зрения. *Компьютерное зрение* — область глубокого обучения, позволяющая машинам видеть, распознавать и обрабатывать изображения, подобно человеку. Зародилась OpenCV в 1999 году как инициатива Intel Research, а теперь поддерживается OpenCV Foundation, некоммерческой организацией, предоставляющей программное обеспечение бесплатно.

Написана эта библиотека на C++, но есть и поддержка других языков, а именно Python и Java. Несмотря на то что в первую очередь она предназначена для приложений компьютерного зрения, работающих в реальном времени, OpenCV

также включает стандартные инструменты по работе с изображениями, подобные используемым в Python Imaging Library. На момент написания книги текущая версия библиотеки — OpenCV 4.1.

Для выполнения числовых и научных вычислений в Python этой библиотеке требуются пакеты Numerical Python (`NumPy`) и `SciPy`. OpenCV рассматривает изображения как трехмерные массивы `NumPy` (рис. 1.4). Это дает ей максимальную функциональную совместимость с другими научными библиотеками Python.

Рис. 1.4. Визуальное представление трехканального массива цветного изображения

OpenCV хранит свойства в виде строк, столбцов и каналов. «Форма» изображения с рис. 1.4 будет выражена кортежем из трех элементов (4, 5, 3). Каждый стек ячеек, например 0-20-40 или 19-39-59, представляет один пиксель. Числа показывают значения интенсивности каждого цветового канала для заданного пикселя.

Поскольку для многих проектов в книге нам потребуются библиотеки `NumPy` и `matplotlib`, сейчас самое время их установить.

Это можно сделать многими способами. Один из них — использование `SciPy`, открытой библиотеки Python, применяемой для научных и технических вычислений (подробнее — на https://scipy.org/index.html).

В качестве альтернативы: если вы собираетесь анализировать множество данных и выполнять чертежи для собственных задач, то можете скачать и использовать бесплатные дистрибутивы Python, например Anaconda или Enthought Canopy, которые работают с Windows, Linux и macOS. Эти дистрибутивы избавят

вас от необходимости поиска и установки нужных версий всех необходимых научных библиотек, таких как NumPy, SciPy и т. д. Список подобных дистрибутивов вместе со ссылками на их сайты можно найти здесь: https://scipy.org/install.html.

Установка NumPy и других научных библиотек с помощью pip

Если вы хотите установить эти продукты напрямую, то используйте *pip (Preferred Installer Program)*, систему управления библиотеками, упрощающую установку ПО для Python (подробнее — на https://docs.python.org/3/installing/). Windows и macOS версии Python, начиная с 3.4, содержат pip по умолчанию. Пользователи Linux могут установить pip отдельно. Для установки или обновления pip обратитесь к инструкциям на странице https://pip.pypa.io/en/stable/installation/ или найдите онлайн-руководство по инсталляции pip для вашей операционной системы.

Я использовал pip для установки научных пакетов, используя инструкции с https://scipy.org/install.html. Так как matplotlib требует несколько зависимостей, их также нужно установить. Для Windows выполните приведенную ниже команду (Python 3) с помощью PowerShell, запущенной (через Shift-правый клик) из каталога с текущей установкой Python:

```
$ python -m pip install --user numpy scipy matplotlib ipython jupyter pandas
sympy nose
```

Если у вас установлены и Python 2, и Python 3, то вместо python используйте python3. Чтобы убедиться в том, что NumPy был установлен и стал доступен для OpenCV, откройте оболочку Python и введите:

```
>>> import numpy
```

Если ошибки не возникнет, то можно устанавливать OpenCV.

Установка OpenCV через pip

Инструкции по установке OpenCV вы найдете на https://pypi.org/project/opencv-python/. Для инсталляции OpenCV в стандартных средах (Windows, macOS и почти всех дистрибутивах GNU/Linux) введите в PowerShell или терминале следующее:

```
pip install opencv-contrib-python
```

либо

```
python -m pip install opencv-contrib-python
```

Если у вас установлено несколько версий Python (например, версии 2.7 и 3.7), то нужно указать ту версию, которую вы хотите использовать.

```
py -3.7 -m pip install --user opencv-contrib-python
```

Если в качестве посредника установки вы используете Anaconda, то можете выполнить эту команду:

```
conda install opencv
```

Чтобы убедиться в корректной установке, введите в оболочке:

```
>>> import cv2
```

Отсутствие ошибок означает, что все в порядке. Если же ошибка возникнет, то обратитесь к списку устранения неполадок по ссылке https://pypi.org/project/opencv-python/.

Код для теоремы Байеса

Программа bayes.py, которую вы напишете в этом разделе, симулирует поиск пропавшего моряка в трех смежных областях. Она будет отображать карту, выводить меню вариантов поиска, произвольно выбирать местоположение моряка и либо показывать его при успешном нахождении, либо выполнять байесовский вывод для вероятностей нахождения в каждой области. Код вместе с изображением карты (cape_python.png) можете скачать с https://nostarch.com/real-world-python/.

Импорт модулей

В листинге 1.1 программа bayes.py начинается с импорта необходимых модулей и присваивания ряда констант. Действия этих модулей мы рассмотрим при их реализации в коде.

Листинг 1.1. Импорт модулей и присваивание констант, используемых в программе bayes.py

bayes.py, part 1

```
import sys
import random
import itertools
import numpy as np
import cv2 as cv

MAP_FILE = 'cape_python.png'
```

```
SA1_CORNERS = (130, 265, 180, 315)  # (UL-X, UL-Y, LR-X, LR-Y)
SA2_CORNERS = (80, 255, 130, 305)  # (UL-X, UL-Y, LR-X, LR-Y)
SA3_CORNERS = (105, 205, 155, 255)  # (UL-X, UL-Y, LR-X, LR-Y)
```

При импорте модулей в программу желательно упорядочить их так: модули стандартной библиотеки (Standard Library) Python, затем сторонние модули, а затем пользовательские. Модуль `sys` включает команды для операционной системы, такие как выход. Модуль `random` позволяет генерировать псевдослучайные числа. Модуль `itertools` помогает в работе с циклами. Наконец, `numpy` и `cv2` импортируют библиотеки `NumPy` и OpenCV соответственно. Также можно присвоить им сокращенные имена (`np`, `cv`) для последующего сокращения написания кода.

Далее выполняется присваивание констант. Согласно руководству по стилю PEP8 (https://www.python.org/dev/peps/pep-0008/), имена констант следует указывать прописными буквами. Это не делает переменные неизменяемыми, но предупреждает других разработчиков, что менять их нельзя.

Карта для вымышленного мыса Python находится в файле изображения cape_python.png (рис. 1.5). Присвойте этот файл постоянной переменной `MAP_FILE`.

Рис. 1.5. Черно-белая карта мыса Python (cape_python.png)

Области поиска будут наноситься на эту карту в виде прямоугольников. OpenCV определит каждый такой прямоугольник по номеру пикселя в его угловых

точках, так что переменную для хранения этих четырех точек нужно создать в виде кортежа. Требуемый порядок: верхний левый *x*, верхний левый *y*, нижний правый *x* и нижний правый *y*. Для представления «области поиска» используйте в имени переменной `SA` (search area).

Определение класса Search

Класс — это тип данных в объектно-ориентированном программировании (ООП). ООП — альтернатива функциональному/процедурному программированию. Оно особенно эффективно для больших сложных программ, так как не только производит код, который проще обновлять, поддерживать и использовать повторно, но также снижает его повторяемость. ООП строится вокруг структур данных, известных как *объекты*, которые состоят из данных, методов и их взаимодействий. В этом качестве оно отлично подходит для игровых программ, где обычно используются взаимодействующие объекты, например космические корабли и астероиды.

Класс — это шаблон, из которого можно создать несколько объектов. Например, у вас может быть класс, создающий линкоры в игре про Вторую мировую войну. Каждый линкор будет наследовать определенные постоянные характеристики, например тоннаж, крейсерскую скорость, уровень топлива, уровень повреждений, вооружение и т. д. При этом каждому объекту линкора также можно задать уникальные характеристики, например индивидуальное имя. После создания, или *инстанцирования*, отдельные характеристики каждого линкора начнут меняться в зависимости от того, сколько топлива сжигается, сколько корабль получает повреждений, сколько использует боеприпасов и т. д.

В bayes.py вы будете использовать класс в качестве шаблона для создания поисково-спасательной миссии, охватывающей три области поиска. В листинге 1.2 определен класс `Search`, который задает схему игры.

Листинг 1.2. Определение класса Search и метода init ()

bayes.py, part 2

```
class Search():
    """Байесовская игра "Поиск и спасение" с 3 областями поиска."""

    def __init__(self, name):
      self.name = name
    ❶ self.img = cv.imread(MAP_FILE, cv.IMREAD_COLOR)
      if self.img is None:
          print('Could not load map file {}'.format(MAP_FILE),
                file=sys.stderr)
          sys.exit(1)

    ❷ self.area_actual = 0
```

```
        self.sailor_actual = [0, 0] # "локальные" координаты в области поиска

    ❸ self.sa1 = self.img[SA1_CORNERS[1] : SA1_CORNERS[3],
                              SA1_CORNERS[0] : SA1_CORNERS[2]]

        self.sa2 = self.img[SA2_CORNERS[1] : SA2_CORNERS[3],
                              SA2_CORNERS[0] : SA2_CORNERS[2]]

        self.sa3 = self.img[SA3_CORNERS[1] : SA3_CORNERS[3],
                              SA3_CORNERS[0] : SA3_CORNERS[2]]

    ❹ self.p1 = 0.2
        self.p2 = 0.5
        self.p3 = 0.3

        self.sep1 = 0
        self.sep2 = 0
        self.sep3 = 0
```

Начнем с определения класса `Search`. Согласно PEP8, первая буква в имени класса должна быть прописной.

Далее определяется метод, устанавливающий начальные значения атрибутов объекта. В ООП *атрибут* — это именованное значение, связанное с объектом. Если объектом является человек, то атрибутом может быть его вес или цвет глаз. *Методы* — это атрибуты, одновременно являющиеся функциями, которым при выполнении передается ссылка на их экземпляр. Метод `_init_()` является особой встроенной функцией, которую Python вызывает автоматически при создании нового объекта. Он привязывает атрибуты каждого создаваемого экземпляра класса. В этом случае передаются два аргумента: `self` и имя, которое вы хотите использовать для объекта.

Параметр `self` — это ссылка на экземпляр создаваемого класса, то есть такого, для которого был вызван метод, технически называемый экземпляром *контекста*. Например, если вы создадите линкор с именем *Missouri*, тогда параметром `self` этого объекта станет `Missouri` и вы сможете вызывать для него метод, например для выстрела из тяжелых орудий, с помощью точечной нотации: `Missouri.fire_big_guns()`. Давая объектам уникальные имена при инстанцировании, вы сохраняете область видимости атрибутов каждого объекта отдельно от других. В этом случае повреждения, полученные одним из линкоров, не повлияют на остальной флот.

Это хорошая практика — перечислять все начальные значения атрибутов объекта под методом `_init_()`. Таким образом, пользователи увидят все ключевые атрибуты объекта, которые будут применяться далее в различных методах, а ваш код станет более читаемым и легким в обновлении. В листинге 1.2 это атрибуты `self`, например `self.name`.

Атрибуты, присвоенные `self`, будут вести себя аналогично глобальным переменным в процедурном программировании. Методы в классе смогут обращаться к ним напрямую, не требуя аргументов. Так как эти атрибуты «ограждены» ширмой *класса*, помех для их использования не возникает, как в случае с реальными глобальными переменными, которые присваиваются внутри глобальной области видимости и изменяются внутри локальных областей отдельных функций.

Далее присваиваем переменную `MAP_FILE` атрибуту `self.img` при помощи метода OpenCV `imread()` ❶. Картинка `MAP_FILE` черно-белая, но вам нужно добавить в нее цвет. Поэтому используем `ImreadFlag` как `cv.IMREAD_COLOR` для загрузки изображения в цветном режиме. Это настроит три цветовых канала (B, G, R) для дальнейшего использования.

Если файла изображения не существует (или пользователь ввел неверное имя файла), OpenCV выдаст не совсем понятную ошибку (`NoneType object is not subscriptable`). Для ее обработки используется проверка условием, которая определяет отсутствие `self.img`, то есть равно ли оно `None`. Если да, то выводится сообщение об ошибке, после чего используется модуль `sys` для выхода из программы. Передача в него кода выхода `1` указывает, что программа завершилась ошибкой. Если установить `file=stder`, то будет использован стандартный красный цвет текста для сообщения об ошибке в окне интерпретатора Python, но не в других окнах, например PowerShell.

Далее присваиваем два атрибута фактического местоположения моряка при его нахождении. Первый хранит количество областей поиска ❷, а второй — точную локацию (x,y). Пока что присвоенные значения будут плейсхолдерами. В дальнейшем мы определим метод для случайного выбора конечных значений. Обратите внимание, что для координат местоположения используется список, поскольку нам нужен изменяемый контейнер.

Изображение карты загружается в виде *массива*. Массив — это коллекция объектов одного типа, имеющая фиксированный размер. Массивы представляют собой эффективные с точки зрения использования памяти контейнеры, обеспечивающие быстрые числовые операции и эффективную логику адресации. Среди концепций, делающих `NumPy` особенно мощным инструментом, можно выделить *векторизацию*. Ее суть — в замене явных циклов более быстрыми выражениями массивов. Как правило, операции применяются к массивам целиком, а не к их отдельным частям. При использовании `NumPy` внутреннее выполнение циклов передается эффективным функциям C или Fortran, которые работают быстрее стандартных техник Python.

Чтобы иметь возможность работать с локальными координатами *внутри* области поиска, мы создаем из массива подмассив ❸. Обратите внимание, что для этого применяется индексация. Сначала предоставляется диапазон от верхнего

левого значения *y* до нижнего правого *y*, а затем от верхнего левого *x* до нижнего правого *x*. Это особенность NumPy. Чтобы привыкнуть к ней, потребуется время, в том числе и потому, что для многих привычнее декартова система координат, где сначала идет *x*, а потом *y*.

Теперь повторим эту процедуру для следующих двух областей поиска, после чего установим предварительные вероятности нахождения моряка в каждой из них ❹. В реальности их бы предоставила программа SAROPS. Очевидно, p1 представляет область 1, p2 — область 2 и т. д. В заключение для SEP устанавливаются атрибуты-плейсхолдеры.

Рисуем карту

Внутри класса Search с помощью функциональности OpenCV мы создадим метод, отображающий базовую карту. Эта карта будет включать области поиска, масштабный отрезок и последнее известное местоположение моряка (рис. 1.6).

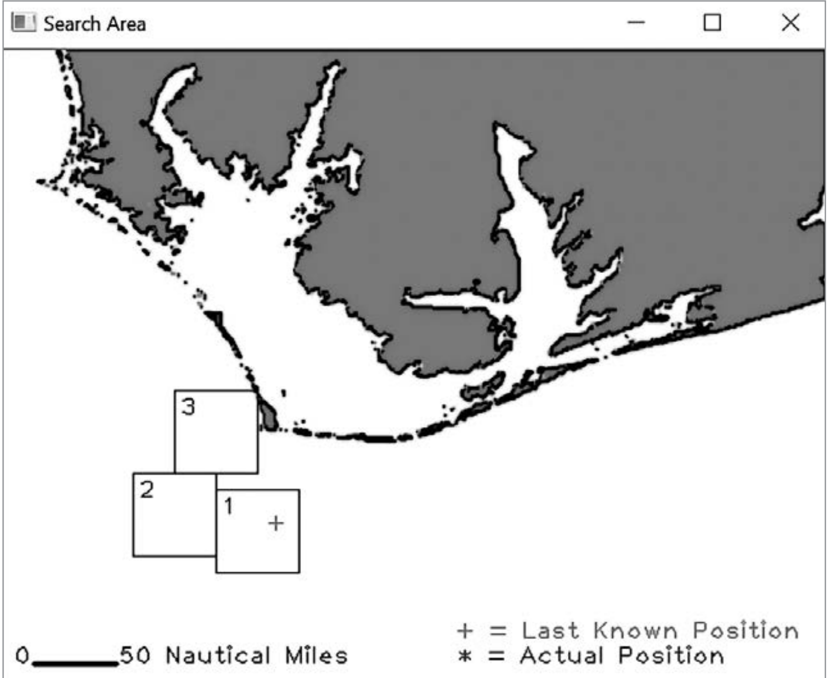


Рис. 1.6. Начальный игровой экран (базовая карта) для bayes.py

В листинге 1.3 определяется метод draw_map(), отображающий начальную карту.

Листинг 1.3. Определение метода для отображения базовой карты

bayes.py, part 3

```
    def draw_map(self, last_known):
        """Отображаем базовую карту с масштабом, последними известными
        координатами xy и областями поиска."""
        cv.line(self.img, (20, 370), (70, 370), (0, 0, 0), 2)
        cv.putText(self.img, '0', (8, 370), cv.FONT_HERSHEY_PLAIN, 1, (0, 0, 0))
        cv.putText(self.img, '50 Nautical Miles', (71, 370),
                   cv.FONT_HERSHEY_PLAIN, 1, (0, 0, 0))

      ❶ cv.rectangle(self.img, (SA1_CORNERS[0], SA1_CORNERS[1]),
                                (SA1_CORNERS[2], SA1_CORNERS[3]), (0, 0, 0), 1)
        cv.putText(self.img, '1',
                   (SA1_CORNERS[0] + 3, SA1_CORNERS[1] + 15),
                   cv.FONT_HERSHEY_PLAIN, 1, 0)
        cv.rectangle(self.img, (SA2_CORNERS[0], SA2_CORNERS[1]),
                     (SA2_CORNERS[2], SA2_CORNERS[3]), (0, 0, 0), 1)
        cv.putText(self.img, '2',
                   (SA2_CORNERS[0] + 3, SA2_CORNERS[1] + 15),
                   cv.FONT_HERSHEY_PLAIN, 1, 0)
        cv.rectangle(self.img, (SA3_CORNERS[0], SA3_CORNERS[1]),
                     (SA3_CORNERS[2], SA3_CORNERS[3]), (0, 0, 0), 1)
        cv.putText(self.img, '3',
                   (SA3_CORNERS[0] + 3, SA3_CORNERS[1] + 15),
                   cv.FONT_HERSHEY_PLAIN, 1, 0)

      ❷ cv.putText(self.img, '+', (last_known),
                   cv.FONT_HERSHEY_PLAIN, 1, (0, 0, 255))
        cv.putText(self.img, '+ = Last Known Position', (274, 355),
                   cv.FONT_HERSHEY_PLAIN, 1, (0, 0, 255))
        cv.putText(self.img, '* = Actual Position', (275, 370),
                   cv.FONT_HERSHEY_PLAIN, 1, (255, 0, 0))

      ❸ cv.imshow('Search Area', self.img)
        cv.moveWindow('Search Area', 750, 10)
        cv.waitKey(500)
```

Определяем метод `draw_map()` с параметрами `self` и `last_known`, указывающими последнее известное местоположение. Далее используем метод OpenCV `line()` для нанесения масштабного отрезка. В качестве аргументов передаем ему изображение базовой карты, кортеж из левых и правых координат (x, y), кортеж цвета отрезка и ширину отрезка.

Используем метод `putText()`, чтобы создать подпись к масштабному отрезку. Передаем ему атрибут для изображения базовой карты, а затем сам текст, сопровождаемый кортежем координат текста в нижнем левом углу. После добавляем имя шрифта, его размер и кортеж цвета.

Теперь рисуем прямоугольник для первой области поиска ❶. Как обычно, передаем изображение базовой карты, далее — переменные, обозначающие четыре

угла рамки, и в завершение — кортеж цвета, а также толщину линии. Снова используем putTest() для размещения номера области поиска внутри ее верхнего левого угла. Повторяем эти шаги для областей поиска 2 и 3.

С помощью putText() размещаем + на последнем известном местоположении моряка ❷. Обратите внимание, что символ красный, но кортеж цвета выглядит как (0, 0, 255), а не (255, 0, 0). Причина в том, что OpenCV использует формат цвета Blue-Green-Red, а не более распространенный Red-Green-Blue (RGB).

Далее размещаем текст, описывающий символы последнего известного местоположения и фактического, которое должно отобразиться в момент обнаружения игроком потерявшегося моряка. Для маркера фактического местоположения используется синий цвет.

Метод завершается показом базовой карты при помощи метода OpenCV imshow() ❸. Ему передается заголовок окна и изображение.

Чтобы окна базовой карты и интерпретатора перекрывали друг друга как можно меньше, определяем расположение базовой карты в верхнем правом углу монитора (возможно, потребуется подстроить координаты для вашего компьютера). Используем метод OpenCV moveWindow()и передаем ему имя окна, ' Search Area', а также координаты левого верхнего угла.

Заканчиваем методом waitKey(), который вносит задержку в *n* мс на время отрисовки изображений в окнах. Передаем ему значение 500, то есть устанавливаем задержку, равную 500 мс. В результате игровое меню должно появляться на полсекунды позже базовой карты.

Выбор фактического местоположения моряка

В листинге 1.4 определяется метод для случайного выбора фактического местоположения моряка. Для удобства координаты изначально выбираются из подмассива области поиска, после чего они преобразуются в глобальные координаты, используемые для всего изображения базовой карты. Этот подход допустим, потому что все области поиска имеют одинаковый размер и форму и, значит, могут использовать одни и те же внутренние координаты.

Листинг 1.4. **Определение метода для случайного выбора фактического местоположения моряка**

bayes.py, part 4

```
    def sailor_final_location(self, num_search_areas):
        """Возвращаем координаты x,y потерявшегося моряка."""
        # Поиск координат моряка в отношении любого подмассива области поиска
          Search Area.
```

```
        self.sailor_actual[0] = np.random.choice(self.sa1.shape[1])
        self.sailor_actual[1] = np.random.choice(self.sa1.shape[0])

    ❶ area = int(random.triangular(1, num_search_areas + 1))

        if area == 1:
            x = self.sailor_actual[0] + SA1_CORNERS[0]
            y = self.sailor_actual[1] + SA1_CORNERS[1]
         ❷ self.area_actual = 1
        elif area == 2:
            x = self.sailor_actual[0] + SA2_CORNERS[0]
            y = self.sailor_actual[1] + SA2_CORNERS[1]
            self.area_actual = 2
        elif area == 3:
            x = self.sailor_actual[0] + SA3_CORNERS[0]
            y = self.sailor_actual[1] + SA3_CORNERS[1]
            self.area_actual = 3
        return x, y
```

Определяем метод `sailor_final_location()` с двумя параметрами: `self` и количеством используемых областей поиска. Для первой координаты (x) в списке `self.sailor.actual` используем метод `random.choice()` из `NumPy`, чтобы выбрать значение из области 1 подмассива. Запомните, области поиска — это массивы `NumPy`, скопированные с более крупного массива изображения. Так как все области поиска/подмассивы имеют один размер, координаты, выбираемые из одного, будут применяться ко всем.

Получить из массива координаты можно с помощью `shape`:

```
>>> print(np.shape(self.SA1))
(50, 50, 3)
```

Атрибут `shape` для массива `NumPy` должен быть кортежем с количеством элементов, соответствующим количеству размерностей массива. При этом нельзя забывать, что для массива в OpenCV элементы кортежа перечисляются в следующем порядке: строки, столбцы, а затем каналы.

Каждая из существующих областей поиска является трехмерным массивом размером 50 × 50 пикселей. Значит, внутренние координаты для x и y имеют значения от 0 до 49. Выбор `[0]` с `random.choice[]` означает использование строк, а заключительный аргумент — `1` — выбирает один элемент. Выбор `[1]` означает столбцы.

Координаты, сгенерированные `random.choice()`, имеют значения от 0 до 49. Чтобы использовать их со всем изображением базовой карты, сначала нужно выбрать область поиска ❶. Для этого используют модуль `random`, который вы

импортировали в начале программы. Согласно выводу SAROPS моряк, скорее всего, находится в области 2 и менее вероятно — в области 3. Поскольку эти начальные целевые вероятности — предположения, которые могут и не соответствовать реальной ситуации, мы выбираем область, где находится моряк, с помощью треугольного распределения. Аргументами являются нижние и верхние конечные точки. Если аргумент для местоположения не предоставлен, то по умолчанию устанавливается средняя точка между конечными значениями. Это согласуется с результатами SAROPS, так как область 2 будет выбираться более часто.

Обратите внимание, что мы используем внутри метода переменную `area`, а не атрибут `self.area`, так как нет необходимости предоставлять к ней доступ из других методов.

Для нанесения местоположения моряка на базовую карту нужно добавить подходящие координаты угловых точек области поиска. Это преобразует «локальные» координаты области поиска в «глобальные» координаты полного изображения базовой карты. Нам также понадобится отслеживать область поиска, так что обновляем атрибут `self.area_actual` ❷.

Повторите эти шаги для областей поиска 2 и 3, а затем верните координаты (x, y).

ПРИМЕЧАНИЕ

В реальной жизни моряк продолжал бы дрейфовать, и шансы его перемещения в область 3 с каждым поиском возрастали бы. Я же предпочел использовать статичное местоположение, чтобы сделать максимально прозрачной логику, стоящую за правилом Байеса. В результате наша игра больше напоминает поиск затонувшей подводной лодки.

Вычисление эффективности поиска и его осуществление

В реальности погодные условия и технические неполадки могут снизить эффективность поиска. Таким образом, стратегия каждого этапа поиска будет такова: генерируем список всех возможных локаций в области, перемешиваем значения списка и далее выбираем результат по значению эффективности поиска. Поскольку SEP никогда не равен 1.0, простая выборка из начала или конца списка — без перемешивания — приведет к тому, что координаты в «хвосте» окажутся невостребованными.

В листинге 1.5 мы в классе `Search` определяем метод для случайного вычисления эффективности заданного поиска и еще один метод для выполнения самого поиска.

Листинг 1.5. Определение методов для случайного выбора эффективности поиска и выполнения самого поиска

bayes.py, part 5

```
    def calc_search_effectiveness(self):
        """Устанавливаем десятичное значение эффективности поиска для каждой
        области поиска."""
        self.sep1 = random.uniform(0.2, 0.9)
        self.sep2 = random.uniform(0.2, 0.9)
        self.sep3 = random.uniform(0.2, 0.9)

❶  def conduct_search(self, area_num, area_array, effectiveness_prob):
        """Возвращаем результаты поиска и список просмотренных координат."""
        local_y_range = range(area_array.shape[0])
        local_x_range = range(area_array.shape[1])
    ❷ coords = list(itertools.product(local_x_range, local_y_range))
        random.shuffle(coords)
        coords = coords[:int((len(coords) * effectiveness_prob))]
    ❸ loc_actual = (self.sailor_actual[0], self.sailor_actual[1])
        if area_num == self.area_actual and loc_actual in coords:
            return 'Found in Area {}.'.format(area_num), coords
    else:
        return 'Not Found', coords
```

Начинаем с метода выбора эффективности поиска. Здесь нужен только один параметр — `self`. Для каждого атрибута эффективности поиска, такого как E1, случайно выбирается значение между 0.2 и 0.9. Это произвольные значения, означающие, что вы всегда будете обыскивать не менее 20 % области, но не более 90 %.

Вы можете заявить, что атрибуты эффективности поиска для наших трех областей являются зависимыми. Туман, к примеру, может повлиять на все три области, повсеместно ухудшая результаты. Вместе с тем, некоторые из вертолетов могут быть оснащены оборудованием для инфракрасного видения, которое позволит улучшить результат. В любом случае, если сделать их независимыми, как делаем мы, симуляция получится более динамической.

Далее мы определяем метод для выполнения поиска ❶. Необходимые параметры здесь — сам объект, номер области (выбираемой пользователем), подмассив этой области и случайно выбранное значение эффективности поиска.

Необходимо сгенерировать список всех координат внутри заданной области поиска. Назовем переменную `local_y_range` и присвоим ей диапазон на основе первого индекса из кортежа с формой массива, который представляет строки. То же самое сделаем для значения `x_range`.

Для генерации списка всех координат в области поиска используем модуль `itertools` ❷. Этот модуль является группой функций в стандартной библиотеке Python, который создает итераторы для эффективного перебора. Функция `product()` возвращает кортежи из всех пермутаций с повторением для заданной

последовательности. В данном случае мы находим все возможные способы совместить x и y в области поиска. Чтобы увидеть это в действии, введите в оболочке следующий сниппет:

```
>>> import itertools
>>> x_range  =  [1,  2, 3]
>>> y_range  =  [4,  5, 6]
>>> coords = list(itertools.product(x_range, y_range))
>>> coords
[(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]
```

Как видите, список `cords` содержит все возможные парные комбинации элементов в списках `x_range` и `y_range`.

Далее идет перемешивание данного списка координат. Это избавит вас от повторяющегося поиска одного и того же конца списка в каждом событии поиска. В следующей строке мы используем срез по индексу, чтобы сократить список на основе вероятности эффективности поиска. Например, низкая эффективность поиска 0.3 означает, что в список включается только одна треть возможных местоположений в области. Поскольку вы будете проверять фактическое местоположение моряка, взяв за основу список, то, по сути, оставите две трети области непросмотренными.

Теперь присваиваем локальную переменную `loc_actual`, которая будет хранить фактическое местоположение моряка ❸. Далее используем условную конструкцию для проверки его нахождения. Если пользователь выбрал верную область поиска, а перемешанный и обрезанный список `cords` содержит местоположение (x, y) моряка, возвращаем строку, сообщающую, что моряк найден, а также список `coords`. В противном случае возвращаем строку, сообщающую, что моряк не найден, и также список `coords`.

Применение байесовского правила и отрисовка меню

Листинг 1.6 также продолжает класс `Search`. Здесь определяются метод и функция. Метод `revise_target_probs()` использует правило Байеса для обновления целевых вероятностей. Они представляют вероятность нахождения моряка в каждой области поиска. Функция `draw_menu()`, определенная вне класса `Search`, отображает меню, которое будет служить графическим интерфейсом пользователя (GUI) для запуска игры.

Листинг 1.6. Определение способов применения теоремы Байеса и отрисовка меню в оболочке Python

bayes.py, part 6

```
    def revise_target_probs(self):
        """Обновляем вероятности целей в области на основе эффективности
        поиска."""
        denom = self.p1 * (1 - self.sep1) + self.p2 * (1 - self.sep2) \
```

```
                + self.p3 * (1 - self.sep3)
        self.p1 = self.p1 * (1 - self.sep1) / denom
        self.p2 = self.p2 * (1 - self.sep2) / denom
        self.p3 = self.p3 * (1 - self.sep3) / denom


def draw_menu(search_num):
    """Выводим меню выбора для проведения поиска в области."""
    print('\nSearch {}'.format(search_num))
    print(
        """
        Choose next areas to search:

        0 - Quit
        1 - Search Area 1 twice
        2 - Search Area 2 twice
        3 - Search Area 3 twice
        4 - Search Areas 1 & 2
        5 - Search Areas 1 & 3
        6 - Search Areas 2 & 3
        7 - Start Over
        """
        )
```

Определяем метод `revise_target_probs()` для обновления вероятности нахождения моряка в каждой области поиска. Единственный параметр здесь — это `self`.

Для удобства мы разбиваем уравнение Байеса на две части, начиная со знаменателя. Нужно умножить предыдущую целевую вероятность на текущее значение эффективности поиска (на с. 29 описано, как это работает).

После вычисления знаменатель используется для завершения уравнения Байеса. В ООП возвращать ничего не нужно. Можно просто обновлять атрибут непосредственно в методе, как если бы это была объявленная глобальная переменная в процедурном программировании.

Далее в глобальной области определяем функцию `draw_menu()` для отрисовки меню. Единственным ее параметром будет номер выполняемого поиска. Так как эта функция не использует `self`, ее не нужно включать в определение класса, хотя это допустимо.

Начинаем с вывода номера поиска. Он потребуется, чтобы выяснить, был ли найден моряк при выполнении разрешенного количества поисков, которое мы установили равным 3.

Для отображения меню используйте с функцией `print()` тройные кавычки. Обратите внимание, что у пользователя есть выбор: отправить обе поисковые группы в одну область или же в разные.

Определение функции main()

Завершив создание класса `Search`, можно наконец привести все эти атрибуты и методы в действие. Листинг 1.7 определяет функцию `main()`, используемую для запуска программы.

Листинг 1.7. Определение начала функции `main()`, используемой для запуска программы

bayes.py, part 7

```
def main():
    app = Search('Cape_Python')
    app.draw_map(last_known=(160, 290))
    sailor_x, sailor_y = app.sailor_final_location(num_search_areas=3)
    print("-" * 65)
    print("\nInitial Target (P) Probabilities:")
    print("P1 = {:.3f}, P2 = {:.3f}, P3 = {:.3f}".format(app.p1, app.p2,
                                                          app.p3))
    search_num = 1
```

Функции `main()` аргументы не требуются. Начнем с создания приложения игры под названием `app`, используя класс `Search`. Назовем этот объект `Cape_Python` (мыс Python).

Далее вызовем метод, отображающий карту. Передадим ему последнее известное местоположение моряка в виде кортежа координат (x, y). Заметьте, что мы используем именованный аргумент, `last_known=(160, 290)`, что вносит некоторую ясность.

Теперь получаем местоположение моряка по осям x и y, вызывая метод, в который передаем количество областей поиска. Далее выводим начальные, или априорные, целевые вероятности, которые были рассчитаны вашими подчиненными из береговой охраны при помощи моделирования методом Монте-Карло, а не правила Байеса. В завершение называем переменную `search_num` и присваиваем ей `1`. Эта переменная отслеживает количество произведенных поисков.

Оценка вариантов меню

Листинг 1.8 начинаем с цикла `while`, используемого для запуска игры в `main()`. Внутри него игрок оценивает и выбирает варианты действий из меню. Можно выбрать: два поиска в одной области, по одному поиску в двух областях, перезапуск игры и выход из нее. Обратите внимание, что игрок может совершать столько поисков, сколько потребуется, чтобы найти моряка; наш трехдневный лимит в игру еще не встроен.

Листинг 1.8. Использование цикла для выбора пунктов меню и запуска игры

bayes.py, part 8

```
    while True:
        app.calc_search_effectiveness()
        draw_menu(search_num)
        choice = input("Choice: ")

        if choice == "0":
            sys.exit()
      ❶ elif choice == "1":
            results_1, coords_1 = app.conduct_search(1, app.sa1, app.sep1)
            results_2, coords_2 = app.conduct_search(1, app.sa1, app.sep1)
          ❷ app.sep1 = (len(set(coords_1 + coords_2))) / (len(app.sa1)**2)
            app.sep2 = 0
            app.sep3 = 0

        elif choice == "2":
            results_1, coords_1 = app.conduct_search(2, app.sa2, app.sep2)
            results_2, coords_2 = app.conduct_search(2, app.sa2, app.sep2)
            app.sep1 = 0
            app.sep2 = (len(set(coords_1 + coords_2))) / (len(app.sa2)**2)
            app.sep3 = 0

        elif choice == "3":
            results_1, coords_1 = app.conduct_search(3, app.sa3, app.sep3)
            results_2, coords_2 = app.conduct_search(3, app.sa3, app.sep3)
            app.sep1 = 0
            app.sep2 = 0
            app.sep3 = (len(set(coords_1 + coords_2))) / (len(app.sa3)**2)

      ❸ elif choice == "4":
            results_1, coords_1 = app.conduct_search(1, app.sa1, app.sep1)
            results_2, coords_2 = app.conduct_search(2, app.sa2, app.sep2)
            app.sep3 = 0

        elif choice == "5":
            results_1, coords_1 = app.conduct_search(1, app.sa1, app.sep1)
            results_2, coords_2 = app.conduct_search(3, app.sa3, app.sep3)
            app.sep2 = 0

        elif choice == "6":
            results_1, coords_1 = app.conduct_search(2, app.sa2, app.sep2)
            results_2, coords_2 = app.conduct_search(3, app.sa3, app.sep3)
            app.sep1 = 0

      ❹ elif choice == "7":
            main()

        else:
            print("\nSorry, but that isn't a valid choice.", file=sys.stderr)
            continue
```

Начнем с цикла `while`, который будет выполняться, пока пользователь не выберет выход из игры. Сразу же используем точечную нотацию для вызова метода, вычисляющего эффективность поиска. Затем вызываем функцию, отображающую игровое меню, и передаем ей номер поиска. Завершаем подготовительную стадию, предлагая пользователю сделать выбор, для чего используем функцию `input()`.

Выбор игрока оцениваем при помощи серии условных выражений. Выбор 0 означает выход из игры. Для выхода используется модуль `sys`, импортированный еще в начале программы.

Если игрок выберет 1, 2 или 3, это будет означать, что он хочет отправить обе поисковые группы в область с соответствующим номером. Вам надо вызвать метод `conduct_search()` дважды, чтобы сгенерировать два набора результатов и координат ❶. Здесь есть нюанс: следует определить общий показатель SEP, поскольку у каждого поиска он будет свой. Для этого мы совмещаем два списка `coords` и преобразуем результат в набор, чтобы удалить все повторы ❷. Получаем длину набора и делим ее на количество пикселей в области 50 × 50. Поскольку другие области мы еще не обыскивали, их SEP устанавливается как `0`.

Повторяем и корректируем предыдущий код для областей поиска 2 и 3. При этом используем выражение `elif`, поскольку в каждом цикле разрешается выбрать только один пункт меню. Это более эффективно, чем использовать дополнительные инструкции `if`, так как выражение `elif`, следующее после ответа `true`, будет пропускаться.

Если игрок выбирает 4, 5 или 6, это означает, что он хочет выполнять поиск по двум областям — по одной команде на область. В этом случае необходимость в пересчете SEP отпадает ❸.

Если игрок хочет начать игру заново, после того как моряка нашли, или просто перезапустить ее в процессе поиска, то вызывается функция `main()` ❹. Она перезапускает игру и очищает карту.

Если игрок делает недопустимый выбор, мы ему об этом сообщаем, после чего используем `continue` для возвращения к началу цикла, где снова просим сделать выбор.

Завершение и вызов main()

Листинг 1.9 продолжает цикл `while`, завершая функцию `main()` и затем вызывая ее для запуска программы.

Листинг 1.9. Завершение и вызов функции main()

bayes.py, part 9

```
            app.revise_target_probs() # формула Байеса для обновления
                                        вероятностей нахождения

            print("\nSearch {} Results 1 = {}"
                  .format(search_num, results_1), file=sys.stderr)
            print("Search {} Results 2 = {}\n"
                  .format(search_num, results_2), file=sys.stderr)
            print("Search {} Effectiveness (E):".format(search_num))
            print("E1 = {:.3f}, E2 = {:.3f}, E3 = {:.3f}"
                  .format(app.sep1, app.sep2, app.sep3))

        ❶ if results_1 == 'Not Found' and results_2 == 'Not Found':
                print("\nNew Target Probabilities (P) for Search {}:"
                      .format(search_num + 1))
                print("P1 = {:.3f}, P2 = {:.3f}, P3 = {:.3f}"
                      .format(app.p1, app.p2, app.p3))
            else:
                cv.circle(app.img, (sailor_x, sailor_y), 3, (255, 0, 0), -1)
              ❷ cv.imshow('Search Area', app.img)
                cv.waitKey(1500)
                main()
            search_num += 1

    if __name__ == '__main__':
        main()
```

Вызываем метод `revise_target_probs()` для применения правила Байеса и повторного вычисления вероятности нахождения моряка в каждой области поиска с учетом его результатов. Далее выводим результаты поиска и вероятность эффективности поиска в оболочке.

Если результаты обоих поисков окажутся отрицательными, отображаем обновленные целевые вероятности, которыми игрок будет руководствоваться при следующем выборе варианта поиска ❶. В противном случае отображаем местоположение найденного моряка на карте. С помощью OpenCV мы рисуем круг и передаем методу изображение базовой карты, кортеж координат моряка (x, y) для текущей точки, радиус (в пикселях), цвет и толщину линии `-1`. Задав отрицательное значение для толщины, мы заполним круг цветом.

Завершаем `main()`, чтобы показать базовую карту, используя код, аналогичный коду в листинге 1.3 ❷. Передаем методу `waitKey()` значение `1500` для отображения фактического местоположения моряка в течение 1.5 с, прежде чем игра вызовет `main()` и автоматически перезапустится. В конце цикла инкрементируем переменную номера поиска на 1. Это нужно делать *после* цикла, чтобы недопустимый выбор не засчитался за поиск.

Возвращаясь в глобальную область, применяем код, который позволяет импортировать программу в виде модуля или запустить в автономном режиме. Переменная `_name` — это встроенная переменная, которую применяют, чтобы установить, является ли программа автономной или импортирована в другую программу. Если вы запускаете эту программу непосредственно, `_name` устанавливается как `_main_`, условие инструкции `if` выполняется и автоматически вызывается `main()`. Если же программа импортирована, то функция `main()` выполняться не будет, пока ее не вызовут намеренно.

Время сыграть

Для запуска игры выберите в текстовом редакторе Run ▸ Run Module или просто нажмите F5. На рис. 1.7 и 1.8 показаны завершающие окна игры с результатами успешного первого поиска.

```
*Python 3.7.2 Shell*
File Edit Shell Debug Options Window Help

Initial Target (P) Probabilities:
P1 = 0.200, P2 = 0.500, P3 = 0.300

Search 1

            Choose next areas to search:

            0 - Quit
            1 - Search Area 1 twice
            2 - Search Area 2 twice
            3 - Search Area 3 twice
            4 - Search Areas 1 & 2
            5 - Search Areas 1 & 3
            6 - Search Areas 2 & 3
            7 - Start Over

Choice: 2

Search 1 Results 1 = Not Found
Search 1 Results 2 = Found in Area 2.

Search 1 Effectiveness (E):
E1 = 0.000, E2 = 0.521, E3 = 0.000
                                                Ln: 436 Col: 0
```

Рис. 1.7. Окно интерпретатора Python с успешным результатом поиска

В этом примере поиска игрок решил отправить обе команды в область 2, начальная вероятность нахождения моряка в которой составляла 50 %. Первый поиск оказался безуспешен, но вторая команда моряка все же нашла. Обратите внимание, что эффективность поиска была всего чуть выше 50 %. Это означает, что шанс обнаружить моряка в первом поиске составлял всего один к четырем

(0.5 × 0.521 = 0.260). Несмотря на разумный выбор, игроку все равно пришлось также положиться и на долю удачи.

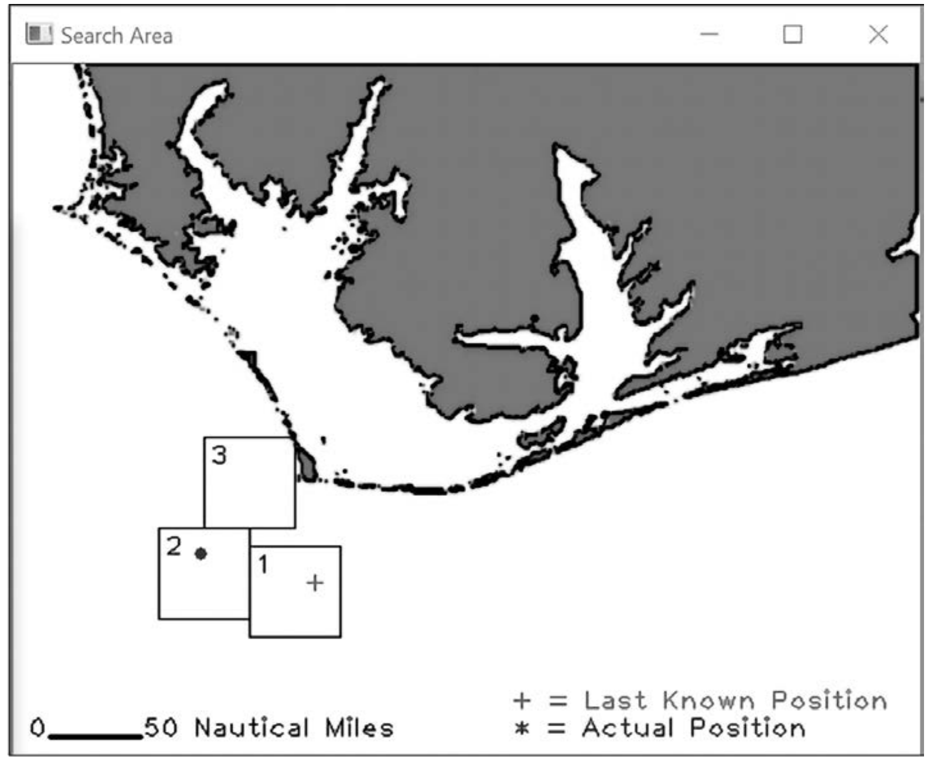


Рис. 1.8. Изображение базовой карты для результата успешного поиска

Когда будете играть в эту игру, попытайтесь погрузиться в сценарий. Ваши решения определяют, будет ли спасен человек или умрет, и времени при этом в обрез. Если моряк дрейфует в воде, то у вас есть всего три попытки, чтобы обнаружить его живым. Используйте их мудро!

Исходя из целевых вероятностей в начале игры, моряк, скорее всего, находится в области 2 и с меньшей долей вероятности — в области 3. Значит, хорошей стратегией начала будет либо дважды произвести поиск по области 2 (выбор меню 2), либо обыскать области 2 и 3 одновременно (выбор меню 6). Вам нужно внимательно следить за выходными данными результатов поиска. Если любая из областей получит высокий показатель эффективности, который означает, что область была просмотрена тщательно, то вам стоит переключить внимание на поиск в других областях.

Вывод ниже представляет одну из худших ситуаций, в которой может оказаться тот, кто принимает решения:

```
Поиск 2 Результаты 1 = Не найден
Поиск 2 Результаты 2 = Не найден

Поиск 2 Эффективность (E):
E1 = 0.000, E2 = 0.234, E3 = 0.610

Новые вероятности местонахождения (P) для поиска 3:
P1 = 0.382, P2 = 0.395, P3 = 0.223
```

После поиска 2, когда остается всего один шанс, целевые вероятности оказываются почти равны, поэтому сложно принять решение о дальнейшем направлении поиска. В этом случае грамотное решение — разделить усилия по двум областям и уповать на лучшее.

Сыграйте несколько раз, вслепую обыскивая области в порядке начальной вероятности, направляя двойные усилия на область 2, затем 3 и, наконец, 1. Затем попробуйте неукоснительно следовать результатам правила Байеса. После этого попробуйте разделить поисковые усилия между областями с двумя наивысшими вероятностями. И в завершение дайте волю своей интуиции, отвергая рекомендации Байеса, если считаете это необходимым. Как это обычно случается, с увеличением областей поиска и дней поиска интуиция очень скоро даст сбой.

Итоги

В этой главе вы изучили правило Байеса, простую статистическую теорему, которая широко применяется и в наше время. Вы написали программу, которая использовала эту теорему для получения новой информации — в форме оценок эффективности поиска — и обновления вероятности нахождения потерянного моряка в каждой исследуемой области.

Вы также скачали и использовали несколько научных пакетов, таких как `NumPy` и OpenCV, которые еще будете неоднократно использовать, изучая материал этой книги. При этом вы применили полезные модули `itertools`, `sys` и `random` из стандартной библиотеки Python.

Дополнительная литература

Книга «The Theory That Would Not Die: How Bayes' Rule Cracked the Enigma Code, Hunted Down Russian Submarines, and Emerged Triumphant from Two Centuries of Controversy» (Yale University Press, 2011), написанная Шэрон Берч Макгрейн (Sharon Bertsch McGrayne), рассказывает об открытии и противоречивости истории теоремы Байеса. В книге есть несколько примеров применения

этой теоремы, один из которых и вдохновил меня на создание сценария с пропавшим моряком.

Обширный источник документации для `NumPy` вы найдете здесь: https://docs.scipy.org/doc/.

Усложняем проект. Более грамотный поиск

На данный момент программа bayes.py помещает все координаты области поиска в список и случайным образом их перетасовывает. В итоге впоследствии могут быть выбраны те варианты поиска в той же области, которые уже были выполнены. Это необязательно плохо в реальной жизни, так как моряк будет постоянно дрейфовать, но в нашем случае лучше охватить максимальный участок без повторов.

Скопируйте и отредактируйте программу, чтобы она отслеживала, области с какими координатами уже были исследованы, и исключала их из будущих поисков (пока снова не будет вызвана `main()`, если моряка обнаружат или игрок выберет вариант 7 в меню для перезапуска). Протестируйте обе версии игры, чтобы увидеть, заметно ли влияют на нее внесенные изменения.

Усложняем проект. Поиск лучшей стратегии с помощью MCS

В моделировании методом Монте-Карло (Monte Carlo simulation, MCS) задействована случайная выборка для прогнозирования результатов в рамках заданных условий. Создайте версию bayes.py, которая автоматически выбирает элементы меню и отслеживает тысячи результатов, позволяя вам определить наиболее эффективную стратегию поиска. К примеру, настройте программу на выбор элементов 1, 2 или 3 в зависимости от наивысшей целевой вероятности по Байесу, а затем запишите номер поиска, который приведет к обнаружению моряка. Повторите эту процедуру 10 000 раз и возьмите среднее по всем номерам поиска. Затем повторите цикл, выбирая элементы меню 4, 5 или 6 в зависимости от наибольшей общей целевой вероятности. Сравните итоговые средние значения. Какой вариант оказался лучше: два поиска в одной области или по одному в каждой?

Усложняем проект. Вычисление вероятности обнаружения

В реальной поисково-спасательной операции вы оценивали бы *ожидаемую* вероятность эффективности поиска для каждой области до выполнения самого поиска. Эта ожидаемая, или *планируемая*, вероятность была бы непосредственно

связана с прогнозом погоды. К примеру, туман смещается в одну область поиска, в то время как в других будет ясно.

Умножение целевой вероятности на планируемый показатель SEP дает *вероятность обнаружения (probability of detection, PoD)* для области. Это вероятность того, что объект будет обнаружен с учетом всех известных ошибок и помех.

Напишите версию bayes.py, которая включает случайно генерируемый планируемый SEP для каждой области поиска. Умножьте целевую вероятность для каждой области (например, `self.p1`, `self.p2` или `self.p3`) на эти новые переменные, чтобы получить PoD для заданной области. К примеру, если целевая вероятность по Байесу для области 3 равна 0.90, а планируемая составляет всего 0.1, то вероятность обнаружения будет 0.09.

Выведите для игрока на экран оболочки целевые вероятности, планируемые SEP и PoD для каждой области, как показано ниже. Теперь игрок сможет использовать эту информацию при принятии решения о дальнейшем поиске.

```
Текущий поиск 1 Эффективность (E):
E1 = 0.190, E2 = 0.000, E3 = 0.000

Новый планируемый поиск Эффективность и вероятность местонахождения (P)
для поиска 2:
E1 = 0.509, E2 = 0.826, E3 = 0.686
P1 = 0.168, P2 = 0.520, P3 = 0.312

Поиск 2

    Выберите область поиска:

    0 - Выход

    1 - Область поиска 1 дважды
      Вероятность обнаружения: 0.164

    2 - Область поиска 2 дважды
      Вероятность обнаружения: 0.674

    3 - Область поиска 3 дважды
      Вероятность обнаружения: 0.382

    4 - Области поиска 1 & 2
      Вероятность обнаружения: 0.515

    5 - Области поиска 1 & 3
      Вероятность обнаружения: 0.3

    6 - Области поиска 2 & 3
      Вероятность обнаружения: 0.643

    7 - Начать заново:
```

Для совмещения PoD при поиске в одной области дважды используйте эту формулу:

$$1 - (1 - PoD)^2.$$

В противном случае суммируйте вероятности.

При вычислении фактического SEP для области ограничьте его ожидаемым значением. При этом будет учитываться общая точность прогнозов погоды, сделанных только на день вперед. Замените метод `random.uniform()` на распределение, например треугольное, построенное на основе планируемого значения SEP. Список доступных видов распределений можно найти на сайте https://docs.python.org/3/library/random.html#real-valued-distributions. Конечно же, фактический показатель SEP для неисследованной области всегда равен 0.

Как внедрение планируемых SEP влияет на геймплей? Выиграть стало проще или сложнее? Насколько сложнее или проще теперь применять теорему Байеса? Если бы вам довелось руководить реальными поисками, то какое решение вы бы приняли в отношении области с высокой целевой вероятностью, но низким SEP из-за бурного волнения на море? Продолжили бы поиск, отозвали бы операцию или перенесли поиск в область с низкой целевой вероятностью, но лучшими погодными условиями?

2

Установление авторства с помощью стилометрии

Стилометрия — это статистический анализ стилистики посредством компьютерного анализа текста. В ее основе лежит тот факт, что у каждого человека есть уникальный, устойчивый и распознаваемый стиль письма. К нему относятся используемый лексикон, нюансы пунктуации, средняя длина предложений и слов, а также другие характеристики.

Одна из распространенных сфер применения стилометрии — установление авторства. Вы когда-нибудь задумывались, а действительно ли именно Шекспир написал все пьесы, принадлежащие, как считается, его перу? Или кто был автором песни «In my life» — Джон Леннон или Пол Маккартни? А может быть, Роберт Гэлбрейт, автор книги «Зов кукушки», — это псевдоним писательницы Дж. К. Роулинг? Стилометрия способна дать ответы на такие вопросы.

Эта техника применяется для снятия обвинений в убийстве и даже помогла вычислить и осудить Унабомбера[1] в 1996 году. Ну и, конечно же, она повсеместно используется для выявления плагиата и определения эмоциональной окраски

---

[1] Американский математик, социальный критик, философ, террорист и неолуддит, известный своей кампанией по рассылке бомб почтой. — *Примеч. ред.*

слов, например в постах в социальных сетях. Стилометрия даже годится для обнаружения признаков депрессии и суицидальных наклонностей.

В этой главе мы с вами при помощи техник стилометрии определим, кто является автором романа «Затерянный мир» — Артур Конан Дойл или Г. Д. Уэллс.

Проект #2: «Собака Баскервилей», «Война миров» и «Затерянный мир»

Сэр Артур Конан Дойл (1859 — 1930) больше всего известен историями о Шерлоке Холмсе, которые считаются эталоном в детективном жанре. Г. Д. Уэллс (1866 — 1946) прославился несколькими новаторскими научно-фантастическими романами — «Война миров», «Машина времени», «Человек-невидимка» и «Остров доктора Моро».

В 1912 году *Strand Magazine* опубликовал «Затерянный мир», серийную версию научно-фантастического романа. Это история экспедиции в бассейн Амазонки под предводительством профессора зоологии Джорджа Эдварда Челленджера, который утверждал, что там водятся динозавры. Собственно, их он там и обнаружил, а также племя злобных обезьяноподобных существ.

Несмотря на то что в реальности автор романа хорошо известен, давайте предположим, что это не так, и наша задача — установить его личность. Эксперты сузили область поиска до двух имен, Дойл и Уэллс. Уэллс выглядит предпочтительнее, потому что «Затерянный мир» относится к жанру научной фантастики, который для этого автора характерен. При этом в книге также встречаются жестокие пещерные люди, напоминающие морлоков из его же романа «Машина времени» (1895). Дойл же, наоборот, известен своими детективными историями и исторической фантастикой.

ЗАДАЧА

Написать на Python программу, использующую стилометрию, чтобы определить, кто является автором романа «Затерянный мир» — Артур Конан Дойл или Г. Д. Уэллс.

Стратегия

Дисциплина *обработка естественного языка (natural language processing, NLP)* занимается взаимодействиями между точным и структурированным языком

компьютеров и специфичным, зачастую неоднозначным «естественным» языком, используемым людьми. NLP применяется для машинного перевода, определения спама, понимания поисковых запросов, а также предиктивного распознавания текста для пользователей сотовых телефонов.

Наиболее типичными тестами NLP на авторство анализируются следующие особенности текста.

- **Длина слов.** График распределения частотности длин слов в документе.
- **Стоп-слова.** График распределения частотности стоп-слов (короткие, внеконтекстные функциональные слова вроде *the*, *but*, *if*).
- **Части речи.** График распределения частотности слов на основе их синтаксической роли (существительные, местоимения, глаголы, обстоятельства, определения и пр.).
- **Наиболее распространенные слова.** Сравнение наиболее часто встречающихся в тексте слов.
- **Коэффициент Жаккара.** Статистика, используемая для оценки сходства и разнообразия выборки.

Если стили письма Дойла и Уэллса отличаются, тогда этих пяти тестов будет достаточно, чтобы выявить их различие. Более подробно о каждом из тестов мы поговорим, когда будем писать код.

Чтобы охватить и проанализировать стиль автора, потребуется образец *корпуса*, иначе говоря — тела текста. Для определения стиля Дойла мы используем известный роман о Шерлоке Холмсе «Собака Баскервилей», опубликованный в 1902 году. Для Уэллса же возьмем книгу «Война миров», выпущенную в 1898 году. Каждый из этих романов содержит более 50 000 слов, чего вполне хватит для репрезентативной статистической выборки. После этого мы сравним выборку каждого автора с произведением *«Затерянный мир»*, чтобы определить, какой стиль для него ближе.

Сам процесс стилометрии будем выполнять с помощью *Natural Language Toolkit (NLTK)*, популярного набора программ и библиотек для работы с данными на естественном языке в Python. Он бесплатный и работает в Windows, macOS, а также Linux. NLTK был разработан в 2001 году в рамках курса компьютерной лингвистики в Университете Пенсильвании и далее развивался усилиями десятков добровольных участников. Более подробно можете узнать об этом проекте на сайте http://www.nltk.org/.

Установка NLTK

Инструкции по установке вы найдете на сайте http://www.nltk.org/install.html. В случае с Windows откройте PowerShell и установите пакет с помощью Preferred Installer Program (pip).

```
python -m pip install nltk
```

Если у вас установлено несколько версий Python, необходимо указать версию. Вот команда для Python 3.7:

```
py -3.7 -m pip install nltk
```

Чтобы убедиться в успешности установки, откройте интерактивную оболочку Python и введите:

```
>>> import nltk
>>>
```

Если ошибок не возникнет, то все в порядке. В противном случае следуйте инструкциям по установке на http://www.nltk.org/install.html.

Скачивание токенизатора

Для выполнения стилометрических тестов потребуется разбивать тексты — их *корпусы* — на отдельные слова, называемые *токенами*. На момент написания книги метод `word_tokenize()` в NLTK неявно вызывает `sent_tokenize()`, используемую для фрагментирования корпуса на отдельные предложения. Для обработки `sent_tokenize()` вам потребуется *Punkt Tokenizer Models*. Несмотря на то что это часть NLTK, нужно скачать и установить ее отдельно с помощью удобного NLTK Downloader. Для его запуска введите в оболочке Python:

```
>>> import nltk
>>> nltk.download()
```

Должно открыться окно NLTK Downloader (рис. 2.1). Перейдите во вкладку Models либо All Packages вверху; затем щелкните punkt в столбце Identifier. Прокрутите окно вниз и установите Download Directory для вашей платформы (см. http://www.nltk.org/data.html). В завершение щелкните на кнопке Download для скачивания Punkt Tokenizer Models.

Заметьте, что пакеты NLTK можно также скачать напрямую из оболочки. Вот пример:

```
>>> import nltk
>>> nltk.download('punkt')
```

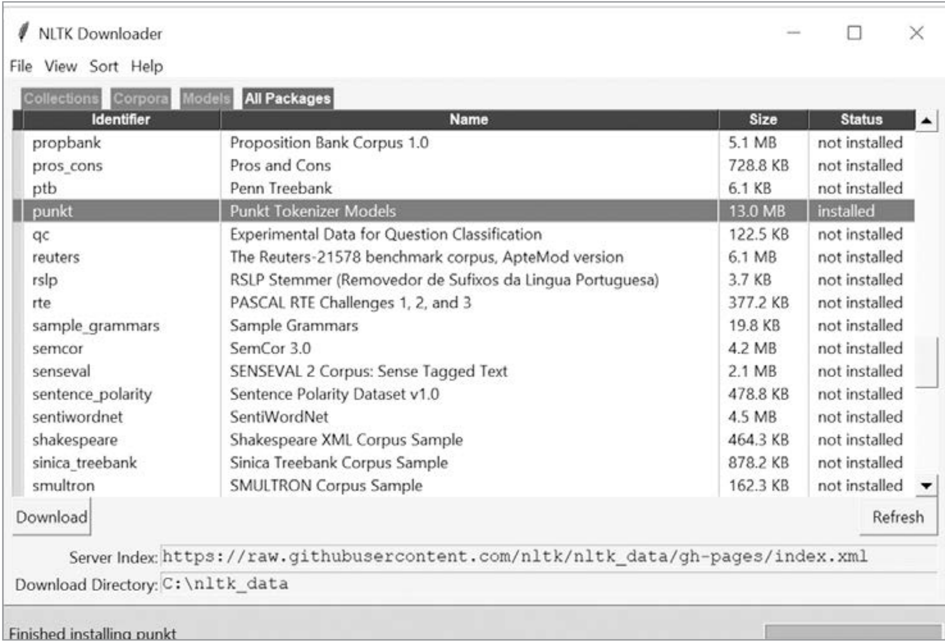


Рис. 2.1. Скачивание Punkt Tokenizer Models

Вам также потребуется обращаться к Stopwords Corpus, который можно скачать тем же способом.

Скачивание Stopwords Corpus

Перейдите во вкладку Corpora в окне NLTK Downloader и скачайте Stopwords Corpus, как показано на рис. 2.2.

В качестве альтернативы можно использовать оболочку.

```
>>> import nltk
>>> nltk.download('stopwords')
```

Давайте скачаем еще один пакет, который поможет анализировать части речи — существительные и глаголы. В окне NLTK Downloader перейдите во вкладку All Packages и скачайте Averaged Perceptron Tagger.

Чтобы сделать это с помощью оболочки, введите:

```
>>> import nltk
>>> nltk.download('averaged_perceptron_tagger')
```

NLTK Downloader

File View Sort Help

Collections | Corpora | Models | All Packages

Identifier	Name	Size	Status
shakespeare	Shakespeare XML Corpus Sample	464.3 KB	not installed
sinica_treebank	Sinica Treebank Corpus Sample	878.2 KB	not installed
smultron	SMULTRON Corpus Sample	162.3 KB	not installed
state_union	C-Span State of the Union Address Corpus	789.8 KB	not installed
stopwords	Stopwords Corpus	17.1 KB	not installed
subjectivity	Subjectivity Dataset v1.0	509.4 KB	not installed
swadesh	Swadesh Wordlists	22.3 KB	not installed
switchboard	Switchboard Corpus Sample	772.6 KB	not installed
timit	TIMIT Corpus Sample	21.2 MB	not installed
toolbox	Toolbox Sample Files	244.7 KB	not installed
treebank	Penn Treebank Sample	1.7 MB	not installed
twitter_samples	Twitter Samples	15.3 MB	not installed
udhr	Universal Declaration of Human Rights Corpus	1.1 MB	not installed
udhr2	Universal Declaration of Human Rights Corpus (Unicode	1.6 MB	not installed
unicode_samples	Unicode Samples	1.2 KB	not installed
universal_treebanks_v20	Universal Treebanks Version 2.0	24.7 MB	not installed

Download Refresh

Server Index: `https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/inde`

Download Directory: `C:\nltk_data`

Рис. 2.2. Скачивание Stopwords Corpus

Когда NLTK завершит скачивание, выйдите из окна NLTK Downloader и введите в интерактивной оболочке Python:

```
>>> from nltk import punkt
```

А затем:

```
>>> from nltk.corpus import stopwords
```

Если ошибки не возникнет, значит, модели и корпус стоп-слов скачаны успешно.

В завершение для построения графиков потребуется `matplotlib`. Если вы ее еще не установили, то смотрите инструкции по установке научных пакетов на с. 32.

Корпусы текстов

Текстовые файлы для книг «Собака Баскервилей» (hound.txt), «Война миров» (war.txt) и «Затерянный мир» (lost.txt), а также код книги доступны на странице https://nostarch.com/real-world-python/.

Они были взяты с Project Gutenberg (https://www.gutenberg.org/), прекрасного источника общедоступной литературы. Чтобы вы могли с ходу начать использовать эти тексты, я убрал из них излишний материал, такой как содержание, названия глав, информация об авторских правах и пр.

Код стилометрии

Программа stylometry.py, которую мы напишем далее, загружает текстовые файлы в виде строк, токенизирует их на слова, после чего выполняет пять стилометрических анализов, перечисленных на с. 59. В качестве результата программа будет выводить набор графиков и сообщений оболочки, которые помогут определить автора книги «Затерянный мир».

Разместите программу в одном каталоге с тремя скачанными текстовыми файлами. Если вы не хотите вводить код самостоятельно, просто загрузите код, используя его доступную для скачивания версию с https://nostarch.com/real-world-python/.

Импорт модулей и определение функции main()

Код листинга 2.1 импортирует NLTK и `matplotlib`, назначает константу и определяет функцию `main()` для запуска программы. Используемые в `main()` функции будут подробно описаны в этой главе позднее.

Листинг 2.1. Импорт модулей и определение функции main()

stylometry.py, часть 1

```
import nltk
from nltk.corpus import stopwords
import matplotlib.pyplot as plt

LINES = ['-', ':', '--'] # Стиль линий для графиков.

def main():
  ❶ strings_by_author = dict()
    strings_by_author['doyle'] = text_to_string('hound.txt')
    strings_by_author['wells'] = text_to_string('war.txt')
    strings_by_author['unknown'] = text_to_string('lost.txt')

    print(strings_by_author['doyle'][:300])

  ❷ words_by_author = make_word_dict(strings_by_author)
    len_shortest_corpus = find_shortest_corpus(words_by_author)
  ❸ word_length_test(words_by_author, len_shortest_corpus)
    stopwords_test(words_by_author, len_shortest_corpus)
    parts_of_speech_test(words_by_author, len_shortest_corpus)
    vocab_test(words_by_author)
    jaccard_test(words_by_author, len_shortest_corpus)
```

Вначале выполняется импорт NLTK и Stopwords Corpus. Далее импортируется `matplotlib`.

После мы создаем переменную `LINES`. По соглашению ее имя прописывается заглавными буквами, это указывает, что она используется в качестве константы. Функция `matplotlib` по умолчанию рисует графики в цвете, но при этом все равно нужно задать список символов для людей, страдающих цветовой слепотой, а также для этой черно-белой книги.

Определяем `main()` и запускаем программу. Шаги в данной функции почти так же читаемы, как псевдокод, и наглядно представляют действия программы. На первом этапе будет выполнена инициализация словаря для хранения текста каждого автора ❶. Функция `text_to_string()` загружает каждый корпус в этот словарь в виде строки. Имя каждого автора будет являться ключом словаря (для «Затерянного мира» используется `unknown`), а строка текста романа — значением. Например, вот ключ `Doyle` с сильно обрезанным строковым значением:

```
{'Doyle': 'Mr. Sherlock Holmes, who was usually very late in the mornings
--snip--'}
```

Сразу после заполнения словаря выводим 300 элементов для ключа `doyle`, чтобы убедиться, что все прошло правильно. На выводе должно получиться следующее:

```
Mr. Sherlock Holmes, who was usually very late in the mornings, save
upon those not infrequent occasions when he was up all night, was seated
at the breakfast table. I stood upon the hearth-rug and picked up the
stick which our visitor had left behind him the night before. It was a
fine, thick piec
```

После корректной загрузки корпусов текстов переходим к токенизации строк в слова. На данный момент Python не распознает слова, он работает с *символами*, такими как буквы, числа и знаки препинания. Чтобы это исправить, мы используем функцию `make_word_dict()`, которая в качестве аргумента будет получать `strings_by_author`, разбивать эти строки на слова и возвращать словарь `words_by_author` с фамилиями авторов в качестве ключей и списком слов в качестве значений ❷.

Стилометрия опирается на подсчет слов, следовательно, лучше всего она работает, когда каждый корпус имеет одинаковую длину. Есть несколько способов обеспечить такое сравнение подобного с подобным. С помощью *разбивки* мы разделим текст на блоки, скажем, по 5000 слов, и сравним эти блоки. Нормализацию можно также производить, используя вместо прямого подсчета относительную частотность или усекая все корпуса по размеру наименьшего из них.

Рассмотрим вариант с усечением. Передадим словарь в другую функцию, `find_shortest_corpus()`, которая вычисляет количество слов в списке каждого

автора и возвращает длину самого короткого корпуса. В табл. 2.1 показана длина каждого корпуса.

Таблица 2.1. Длина (количество слов) каждого корпуса

Корпус	Длина
Hound (Doyle)	58,387
War (Wells)	59,469
World (Unknown)	74,961

Поскольку наименьший корпус здесь представляет робастный датасет почти из 60 000 слов, мы используем переменную `len_shortest_path`, чтобы обрезать другие два корпуса до этой длины, и уже затем перейдем к анализу. При этом мы, конечно же, предполагаем, что обрезаемое содержание текстов не сильно отличается от оставляемого.

На следующих пяти строках вызываются функции, выполняющие стилометрический анализ, представленный в разделе «Стратегия» на с. 58. ❸ Все эти функции получают в качестве аргумента словарь `words_by_author`, и большая их часть также получает `len_shortest_corpus`. Данные функции мы рассмотрим, как только закончим подготовку текстов к анализу.

Загрузка текста и построение словаря слов

В листинге 2.2 определяются две функции. Первая считывает текстовый файл в виде строки. Вторая создает словарь с именем каждого автора в качестве ключа и его романом, токенизированным из непрерывной строки в отдельные слова, в качестве значения.

Листинг 2.2. Определение функций text_to_string() и make_word_dict()

stylometry.py, часть 2

```
   def text_to_string(filename):
       """Читаем текстовый файл и возвращаем строку."""
       with open(filename) as infile:
           return infile.read()

❶ def make_word_dict(strings_by_author):
       """Возвращаем словарь слов-токенов корпусов по автору."""
       words_by_author = dict()
       for author in strings_by_author:
           tokens = nltk.word_tokenize(strings_by_author[author])
         ❷ words_by_author[author] = ([token.lower() for token in tokens
                                       if token.isalpha()])
       return words_by_author
```

Сначала определяется функция `text_to_string()`, загружающая текстовый файл. Встроенная функция `read()` считывает весь файл как отдельную строку, позволяя выполнять относительно простые манипуляции со всем его содержимым. Для открытия файла используется `with`, что гарантирует его закрытие вне зависимости от того, как завершится блок. Закрыть за собой файл — это все равно что собрать с пола игрушки после игры. Эта практика исключает вероятность неприятных ситуаций, таких как исчерпание файловых дескрипторов, блокировка файла для дальнейшего доступа, повреждение файлов или утрата данных при записи в них.

Некоторые пользователи при загрузке текста могут столкнуться с ошибкой `UnicodeDecodeError`, наподобие такой:

```
UnicodeDecodeError: 'ascii' codec can't decode byte 0x93 in position 365:
ordinal not in range(128)
```

Кодирование и *декодирование* означает процесс преобразования символов, хранящихся в виде байтов, в понятные человеку строки. Проблема в том, что предустановленное кодирование для встроенной функции `open()` платформозависимо и определяется значением `locale.getpreferredencoding()`. Например, при выполнении под Windows 10 вы получите следующее кодирование:

```
>>> import locale
>>> locale.getpreferredencoding()
'cp1252'
```

CP-1252 — это устаревшая кодировка символов в Windows. Если выполнить тот же код на Mac, то может вернуться нечто другое, например `'US-ASCII'` или `'UTF-8'`.

UTF *(Unicode Transformational Format)*, или формат преобразования Юникода, — формат текстовых символов, разработанный для поддержки совместимости с ASCII. Несмотря на то что UTF-8 может обрабатывать все наборы символов и является доминирующей формой кодирования, используемой в мировой сети, — по умолчанию во многих текстовых редакторах он не используется.

Кроме того, в Python 2 предполагалось, что все текстовые файлы кодируются с помощью `latin-1`, используемой для латинского алфавита. Python 3 уже мудрее и пытается обнаружить проблемы с кодировкой как можно раньше. Тем не менее, если кодировка не задана, он может выдать ошибку.

Итак, первый шаг по решению проблемы состоит в передаче `open()` аргумента `encoding` с указанием UTF-8.

```
with open(filename, encoding='utf-8') as infile:
```

Если у вас по-прежнему возникают сложности с загрузкой файлов корпусов, попробуйте добавить аргумент `errors`:

```
with open(filename, encoding='utf-8', errors='ignore') as infile:
```

Ошибки можно игнорировать, потому что эти текстовые файлы были скачаны как UTF-8 и уже проверены с помощью этого подхода. Более подробно о UTF-8 можете почитать на https://docs.python.org/3/howto/unicode.html.

Далее определяем функцию `make_word_dict()`, которая будет по имени автора получать словарь строк и возвращать словарь слов ❶. Сначала инициализируем пустой словарь `words_by_author`. Затем перебираем ключи в словаре `strings_by_author`. Используем метод NLTK `word_tokenize()` и передаем ему ключ словаря строк. В результате получим список токенов, которые будут служить в качестве значения словаря для каждого автора. Токены — это просто нарезанные фрагменты корпуса, как правило, предложения или слова.

Следующий сниппет демонстрирует, как непрерывная строка преобразуется в список токенов (слов и знаков препинания):

```
>>> import nltk
>>> str1 = 'The rain in Spain falls mainly on the plain.'
>>> tokens = nltk.word_tokenize(str1)
>>> print(type(tokens))
<class 'list'>
>>> tokens
['The', 'rain', 'in', 'Spain', 'falls', 'mainly', 'on', 'the', 'plain', '.']
```

Это похоже на использование встроенной в Python функции `split()`, но `split()` не получает токены с лингвистической точки зрения (заметьте, что точка не токенизируется).

```
>>> my_tokens = str1.split()
>>> my_tokens
['The', 'rain', 'in', 'Spain', 'falls', 'mainly', 'on', 'the', 'plain.']
```

После получения токенов заполняем словарь `words_by_author` с помощью спискового включения (list comprehension) ❷. *Списковое включение* — быстрый способ выполнения циклов в Python. Чтобы обозначить список, нужно заключить код в квадратные скобки. Преобразуем токены в нижний регистр и используем встроенный метод `isalpha()`, который возвращает `True`, если все символы в токене являются частью алфавита, и `False` в противном случае. Так мы отфильтруем числа и знаки препинания. Это также исключит слова с дефисами и имена. Завершаем процесс возвращением словаря `words_by_author`.

Поиск самого короткого корпуса

В компьютерной лингвистике *частотность* означает количество вхождений в корпусе. Таким образом, частотность — это *количество*, и методы, которые вы далее будете использовать, возвращают словарь слов и их количество. Чтобы сравнить количество значимым образом, все корпусы должны иметь одинаковое количество слов.

Поскольку три используемых в нашем случае корпуса достаточно велики (см. табл. 2.1), можно безопасно нормализовать их, обрезав до длины самого короткого. В листинге 2.3 определяется функция, которая находит самый короткий корпус в словаре `words_by_author` и возвращает его длину.

Листинг 2.3. Определение функции find_shortest_corpus()

stylometry.py, часть 3

```
def find_shortest_corpus(words_by_author):
    """Вернуть длину самого короткого корпуса."""
    word_count = []
    for author in words_by_author:
        word_count.append(len(words_by_author[author]))
        print('\nNumber of words for {} = {}\n'.
              format(author, len(words_by_author[author])))
        len_shortest_corpus = min(word_count)
        print('length shortest corpus = {}\n'.format(len_shortest_corpus))
        return len_shortest_corpus
```

В начале определяем функцию, получающую в качестве аргумента словарь `words_by_author`, и сразу создаем пустой список для подсчета слов.

Далее перебираем имена авторов (ключи) в словаре. Получаем длину значения для каждого ключа, являющегося объектом-списком, и прибавляем длину в список `word_count`. Здесь длина представляет количество слов в корпусе. Для каждого прохода цикла выводим имя автора и длину токенизированного корпуса.

Когда цикл завершается, используем встроенную функцию `min()` для получения наименьшего количества слов и присваиваем его переменной `len_shortest_corpus`. Выводим ответ, после чего возвращаем эту переменную.

Сравнение длины слов

Одна из особенностей стиля каждого автора — его словарный запас. Фолкнер заметил, что Хемингуэй никогда не заставлял читателя прибегать к помощи словаря; а Хемингуэй обвинил Фолкнера в использовании «10-долларовых слов». Авторский стиль определяется также длиной слов и характерным лексиконом; это мы рассмотрим немного позже.

В листинге 2.4 определяется функция сравнения длины слов для каждого корпуса и построения графика результатов в виде распределения частотности. В распределении частотности длина слов отражается согласно количеству вхождений слов каждой длины. К примеру, слова в шесть букв у одного автора могут встречаться 4000 раз, а у другого — 5500. Распределение частотности позволяет проводить сравнение по диапазонам различных длин слов, а не по усредненному значению длины.

Функция листинга 2.4 с помощью среза уменьшает списки слов до длины самого короткого корпуса, чтобы результаты не искажались размером романа.

Листинг 2.4. Определение функции word_length_test()

stylometry.py, part 4

```
def word_length_test(words_by_author, len_shortest_corpus):
    """Распределение частотности длины слов в корпусах по автору, по самому
    короткому корпусу."""
    by_author_length_freq_dist = dict()
    plt.figure(1)
    plt.ion()

  ❶ for i, author in enumerate(words_by_author):
        word_lengths = [len(word) for word in words_by_author[author]
                        [:len_shortest_corpus]]
        by_author_length_freq_dist[author] = nltk.FreqDist(word_lengths)
      ❷ by_author_length_freq_dist[author].plot(15,
                                                linestyle=LINES[i],
                                                label=author,
                                                title='Word Length')

    plt.legend()
    #plt.show() # Раскомментировать (удалить #) для просмотра графика
                  во время кодирования.
```

Все стилометрические функции обращаются к словарю токенов. Почти во всех используется параметр длины самого короткого корпуса, чтобы обеспечить согласованность размеров образцов текстов. Эти имена переменных мы задействуем в качестве параметров функций.

Сначала создаем пустой словарь для хранения распределения частотности длин слов по авторам, затем чертим графики. Поскольку графиков будет несколько, сначала инстанцируем объект фигуры `1`. Чтобы графики не исчезли после создания, включаем интерактивный режим с помощью `plt.ion()`.

Далее перебираем авторов в токенизированном словаре ❶. Посредством функции `enumerate()` генерируем индекс для каждого автора, который будем использовать для определения стиля линии графика. Для каждого автора применяем списковое включение, чтобы получить длину каждого слова в списке значений, диапазон

которого уменьшен до длины самого короткого корпуса. В результате получим список, где каждое слово заменено целым числом, показывающим его длину.

Теперь начинаем заполнять новый словарь по авторам распределениями частотностей. Здесь мы используем `nltk.FreqDist()`, получающую список длин слов и создающую объект данных с информацией о частотности слов, которую мы отразим на графике.

Словарь покажем на графике с помощью метода класса `plot()`, не ссылаясь на `pyplot` через `plt` ❷. Таким образом, мы сначала отразим на графике наиболее часто встречающийся образец, сопровождаемый количеством заданных образцов, в данном случае `15`. Это означает, что мы увидим распределение частотности слов длиной от 1 до 15 букв. Далее используем `i` для выборки из списка `LINES` и завершаем предоставлением метки и названия. Метку мы используем в легенде, вызываемой с помощью `plt.legend()`.

Заметьте, что можно изменить способ формирования графика распределения частотности при помощи параметра `cumulative`. Если вы установите `cumulative=True`, то увидите кумулятивное распределение (рис. 2.3, слева). В противном случае `plot()` по умолчанию использует `cumulative=False`, и вы увидите фактическое количество вхождений, упорядоченных от большего к меньшему (рис. 2.3, справа). Для данного проекта продолжим использовать вариант по умолчанию.

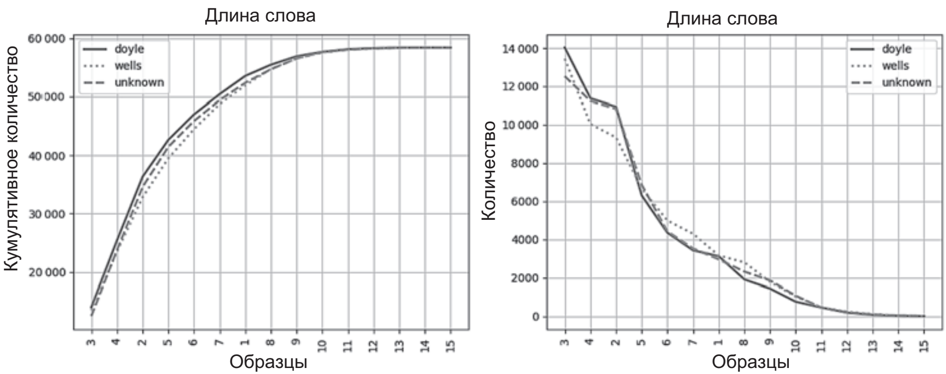


Рис. 2.3. Кумулятивный график NLTK (слева) и распределение частотности по умолчанию (справа)

Вызываем метод `plt.show()` для отображения графика, но пока оставляем его закомментированным. Если вы захотите сразу же увидеть график, то можете раскомментировать метод. Обратите внимание, что при запуске этой программы через Windows PowerShell графики могут закрываться сразу, пока вы не

установите флаг `block`: `plt.show(block=True)`. Это поддержит график открытым, но приостановит выполнение программы до момента его закрытия.

Если проанализировать только график частотности длин слов на рис. 2.3, то стиль Дойла совпадает со стилем неизвестного автора больше, хотя есть отрезки, где стиль Уэллса соответствует так же или даже лучше. Посмотрим, что покажут другие тесты.

Сравнение стоп-слов

Стоп-слово — это короткое, часто используемое слово, например *the, by* и *but*. При выполнении задач вроде онлайн-поиска эти слова отфильтровываются, так как не несут контекстной информации; для определения авторства они также малозначимы.

Однако стоп-слова, а они используются часто и без особого умысла, являются одним из лучших признаков авторского стиля. А поскольку сравниваемые тексты обычно относятся к разным тематикам, то эти стоп-слова становятся важными, так как не привязаны к содержанию и используются в любом тексте.

В листинге 2.5 определяется функция сравнения использования стоп-слов в наших трех корпусах.

Листинг 2.5. Определение функции stopwords_test()

`stylometry.py`, часть 5

```
def stopwords_test(words_by_author, len_shortest_corpus):
    """График частотности стоп-слов в корпусах по автору, по самому короткому
    корпусу"""
    stopwords_by_author_freq_dist = dict()
    plt.figure(2)
    stop_words = set(stopwords.words('english')) # Используем множество
                                                   для скорости
    #print('Number of stopwords = {}\n'.format(len(stop_words)))
    #print('Stopwords = {}\n'.format(stop_words))

    for i, author in enumerate(words_by_author):
        stopwords_by_author = [word for word in words_by_author[author]
                              [:len_shortest_corpus] if word in stop_words]
        stopwords_by_author_freq_dist[author] = nltk.FreqDist(stopwords_by_
        author)
        stopwords_by_author_freq_dist[author].plot(50,
                                                  label=author,
                                                  linestyle=LINES[i],
                                                  title=
                                                  '50 Most Common Stopwords')
    plt.legend()
##    plt.show() # Раскомментируйте, чтобы видеть график в процессе
                 написания функции.
```

Определяем функцию, получающую в качестве аргументов переменные словаря слов и длины самого короткого корпуса. Далее инициализируем словарь для хранения распределения частотности стоп-слов для каждого автора. Чертить все графики на одном рисунке — не самая лучшая идея, поэтому мы создадим новый рисунок под номером `2`.

Присваиваем локальную переменную `stop_words` корпусу стоп-слов NLTK для английского языка. По множествам поиск происходит быстрее, чем по спискам, поэтому делаем корпус множеством для дальнейшего ускорения поиска по нему. Следующие две строки, пока закомментированные, выводят количество стоп-слов (179) и сами стоп-слова.

Перебираем авторов в словаре `words_by_author`. С помощью спискового включения выбираем все стоп-слова в корпусе каждого автора и используем их в качестве значения в новом словаре `stopwords_by_author`. В следующей строке передаем этот словарь NLTK методу `FreqDist()` и используем вывод для заполнения словаря `stopwords_by_author_freq_dist`. Он будет содержать данные, необходимые для создания графиков распределения частотности для каждого автора.

Повторяем код, использованный для отрисовки графика длин слов в листинге 2.4, но количество образцов устанавливаем равным 50 и даем ему другое имя. Таким образом, мы построим график для 50 наиболее часто используемых стоп-слов (рис. 2.4).

Дойл и неизвестный автор используют стоп-слова похожим образом. На этот момент два проведенных теста указывают на Дойла как на более вероятного автора неизвестного текста, но окончательные выводы делать рано.

Сравнение частей речи

Теперь сравним используемые в рассматриваемых корпусах части речи. NLTK задействует для распознавания разметчик частей речи (part-of-speech, POS), называемый `PerceptronTagger`. Разметчики POS обрабатывают последовательность токенизированных слов и прикрепляют тег POS к каждому слову (табл. 2.2).

Разметчики, как правило, обучаются на больших датасетах вроде *Penn Treebank* или *Brown Corpus*, что делает их намного более точными, но все же несовершенными. Также можно найти обучающие данные и разметчики для других языков. Тем не менее вам не стоит озадачиваться всеми этими терминами и их сокращениями. Как и в предыдущих тестах, здесь вам понадобится только сравнить линии на графиках.

В листинге 2.6 определяется функция построения графика распределения частотности POS в трех корпусах.

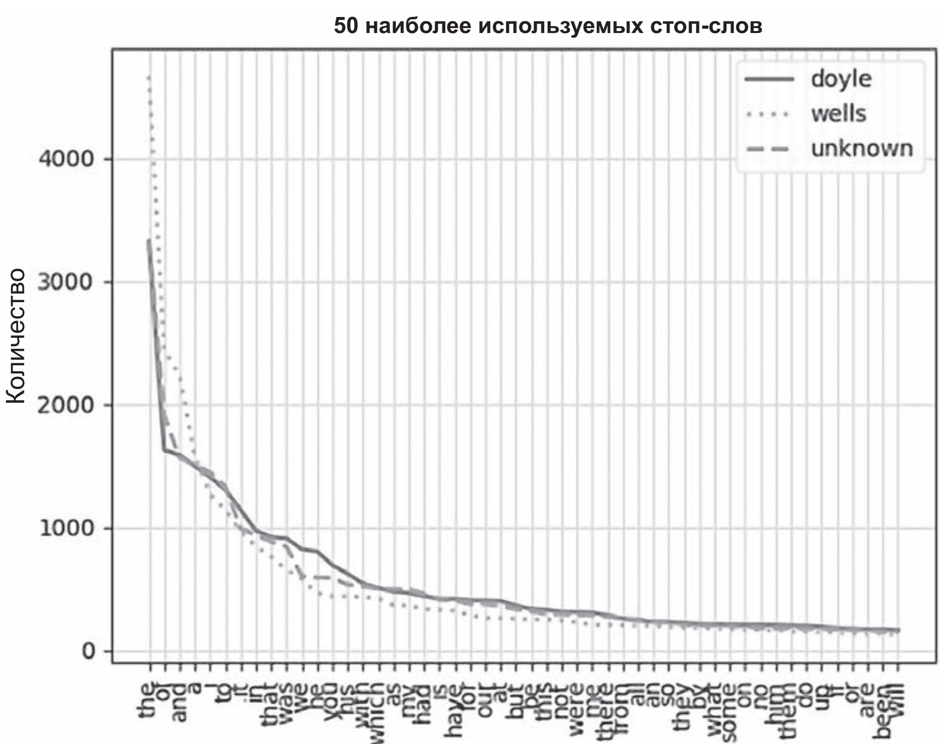


Рис. 2.4. График частотности 50 самых используемых стоп-слов каждым автором

Таблица 2.2. Части речи со значениями тегов

Часть речи	**Тег**	**Часть речи**	**Тег**
Сочинительный союз	CC	Притяжательное местоимение	PRP$
Кардинальное число	CD	Наречие	RB
Определяющее слово	DT	Наречие, компаратив	RBR
Оборот с there	EX	Наречие, суперлатив	RBS
Иностранное слово	FW	Частица	RP
Предлог или подчинительный союз	IN	Символ	SYM
Прилагательное	JJ	Частица to	TO
Прилагательное, компаратив	JJR	Междометие	UH
Прилагательное, суперлатив	JJS	Глагол, базовая форма	VB

Таблица 2.2 (*окончание*)

Часть речи	Тег	Часть речи	Тег
Маркер элементов списка	LS	Глагол, прошедшее время	VBD
Модальность	MD	Глагол, герундий или причастие настоящего времени	VBG
Существительное, единственное или множественное	NN	Глагол, причастие прошедшего времени	VBN
Существительное, множественное	NNS	Глагол не третьего лица единственного числа настоящего времени	VBP
Существительное, имя собственное, единственное число	NNP	Глагол третьего лица единственного числа настоящего времени	VBZ
Существительное, имя собственное, множественное число	NNPS	Wh-определитель, which	WDT
Предетерминатив	PDT	Wh-местоимение, who, what	WP
Притяжательное окончание	POS	Притяжательное wh-местоимение, whose	WP$
Личное местоимение	PRP	Wh-наречие, where, when	WRB

Листинг 2.6. Определение функции parts_of_speech_test()

stylometry.py, часть 6

```
def parts_of_speech_test(words_by_author, len_shortest_corpus):
    """Нарисуем график использования автором разных частей речи"""
    by_author_pos_freq_dist = dict()
    plt.figure(3)
    for i, author in enumerate(words_by_author):
        pos_by_author = [pos[1] for pos in nltk.pos_tag(words_by_author[author]
                         [:len_shortest_corpus])]
        by_author_pos_freq_dist[author] = nltk.FreqDist(pos_by_author)
        by_author_pos_freq_dist[author].plot(35,
                                             label=author,
                                             linestyle=LINES[i],
                                             title='Part of Speech')
    plt.legend()
    plt.show()
```

Определяем функцию, получающую в качестве аргументов опять же словарь слов и длину самого короткого корпуса. Далее инициализируем словарь для хранения распределения частотности POS для каждого автора, после чего вызываем функцию для создания третьего рисунка.

Начинаем перебирать авторов в словаре `words_by_author` и используем списковое включение с методом NLTK `pos_tag()` для построения списка `pos_by_author`. Таким образом, для каждого автора будет создан список, в котором любое слово его корпуса будет заменено соответствующим тегом POS, как показано здесь:

```
['NN', 'NNS', 'WP', 'VBD', 'RB', 'RB', 'RB', 'IN', 'DT', 'NNS', --snip--]
```

Далее вычисляем распределение частотности списка POS и с каждым циклом чертим кривую на основе 35 образцов. Заметьте, что всего существует 36 тегов POS и некоторые, например *маркеры элементов списка*, встречаются в романах редко.

Это последний из наших графиков, поэтому вызываем `plt.show()` для его вывода на экран. Как я уже говорил при рассмотрении листинга 2.4, если вы запускаете программу через Windows PowerShell, то есть вероятность, что вам потребуется использовать `plt.show(block=True)`, чтобы избежать автоматического закрытия графиков.

Предыдущие графики, как и текущий (рис. 2.5), должны появляться спустя примерно 10 секунд.

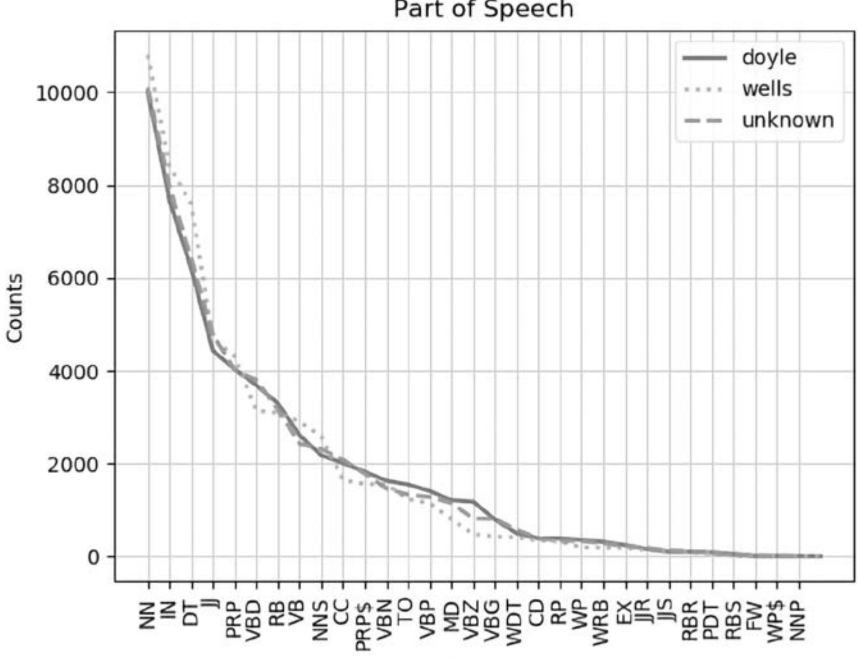


Рис. 2.5. График частотности 35 наиболее используемых каждым автором частей речи

И снова кривая неизвестного автора больше соответствует кривой Дойла, нежели Уэллса. Это указывает на то, что корпус неизвестного автора принадлежит перу Дойла.

Сравнение лексикона авторов

Для сравнения лексиконов трех корпусов мы будем использовать *случайную величину, распределенную по закону хи-квадрат* (X^2), также называемую *статистическим критерием*. На ее основе мы измерим «расстояния» между лексиконами, используемыми в корпусе неизвестного автора и в корпусах известных. Самые близкие лексиконы окажутся наиболее схожими.

$$X^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}.$$

Здесь O — это наблюдаемое количество слов, а E — это ожидаемое их количество при условии, что сравниваемые корпусы относятся к одному автору.

Если оба романа написал Дойл, то в обоих доля наиболее часто используемых слов должна быть одинаковой или схожей. Статистический критерий позволяет квантифицировать степень их схожести посредством измерения количественной разницы использования каждого слова. Чем ниже статистический критерий хи-квадрат, тем больше сходство двух распределений.

В листинге 2.7 определяется функция для сравнения лексиконов трех корпусов.

Листинг 2.7. Определение функции vocab_test()

stylometry.py, часть 7

```
def vocab_test(words_by_author):
    """Сравнение лексиконов авторов на основе статистического теста
    хи-квадрат"""
    chisquared_by_author = dict()
    for author in words_by_author:
      ❶ if author != 'unknown':
            combined_corpus = (words_by_author[author] +
                               words_by_author['unknown'])
            author_proportion = (len(words_by_author[author])/
                                 len(combined_corpus))
            combined_freq_dist = nltk.FreqDist(combined_corpus)
            most_common_words = list(combined_freq_dist.most_common(1000))
            chisquared = 0
          ❷ for word, combined_count in most_common_words:
                observed_count_author = words_by_author[author].count(word)
                expected_count_author = combined_count * author_proportion
                chisquared += ((observed_count_author -
                               expected_count_author)**2 /
                              expected_count_author)
```

```
            ❸ chisquared_by_author[author] = chisquared
            print('Chi-squared for {} = {:.1f}'.format(author, chisquared))
    most_likely_author = min(chisquared_by_author, key=chisquared_by_author.get)
    print('Most-likely author by vocabulary is {}\n'.format(most_likely_author))
```

Функции `vocab_test()` требуется словарь слов, на этот раз без длины самого короткого корпуса. Хотя, как и в предыдущих случаях, она сначала создает новый словарь для хранения значения хи-квадрат для каждого автора, после чего перебирает словарь слов.

Для вычисления хи-квадрата нужно объединить корпус каждого автора с корпусом неизвестного автора. Нам не нужно совмещать `unknown` с самим собой, для чего используем условную конструкцию ❶. Для текущего цикла совмещаем корпус автора с неизвестным корпусом, а затем получаем пропорцию слов для данного автора путем деления длины его корпуса на длину совмещенного корпуса. Далее получаем распределение частотности совмещенного корпуса, вызвав `nltk.FreqDist()`.

Теперь создаем список из 1000 самых распространенных слов в совмещенном тексте, для чего используем метод `most_common()`, которому передаем значение `1000`. Точного критерия для количества рассматриваемых в стилометрическом анализе слов не существует. В литературе чаще всего предполагается использование числа в диапазоне от 100 до 1000. А раз мы работаем с большими текстами, то и предпочтение отдаем большему значению.

Инициализируем переменную `chisquared` с `0`, затем запускаем вложенный цикл `for`, который перерабатывает список `most_common_words` ❷. Метод `most_common()` возвращает список кортежей, каждый из которых содержит слово и количество его вхождений.

```
[('the', 7778), ('of', 4112), ('and', 3713), ('i', 3203), ('a', 3195),
--snip--]
```

Далее для каждого автора мы получаем найденное количество слов из словаря. Для Дойла это будет количество наиболее употребляемых слов в корпусе «Собака Баскервилей». Затем получаем ожидаемое количество слов для Дойла, если бы «Собаку Баскервилей» и неизвестный корпус написал он. Для этого умножаем количество вхождений в совмещенном корпусе на ранее вычисленную пропорцию для автора. Далее применяем формулу хи-квадрат и добавляем результат в словарь, отслеживающий показатель хи-квадрат каждого автора ❸. В итоге получим результат для каждого автора.

Чтобы найти автора с наименьшим показателем хи-квадрат, вызываем встроенную функцию `min()` и передаем ей словарь вместе с ключом словаря, который возвращается из метода `get()`. Так мы получим *ключ*, соответствующий минимальному *значению*. Это важно. Если опустить последний аргумент, `min()` вернет

минимальный *ключ* на основе алфавитного порядка имен, а *не* их показателя хи-квадрат. Подобная ошибка показана в следующем сниппете:

```
>>> print(mydict)
{'doyle': 100, 'wells': 5}
>>> minimum = min(mydict)
>>> print(minimum)
'doyle'
>>> minimum = min(mydict, key=mydict.get)
>>> print(minimum)
'wells'
```

Легко предположить, что функция `min()` возвращает минимальное численное *значение*, но, как вы видите, по умолчанию оно выглядит как *ключи* словаря.

Завершите функцию выводом имени наиболее вероятного автора, исходя из показателя хи-квадрат.

```
Chi-squared for doyle = 4744.4 Chi-squared for wells = 6856.3
Наиболее вероятным по лексикону автором является doyle
```

И еще один тест показал, что автором является Дойл!

Вычисление коэффициента Жаккара

Для определения степени сходства между созданными из корпусов множествами мы будем использовать *коэффициент сходства Жаккара*, также называемый *пересечением относительно объединения* (intersection over union). Это просто область пересечения двух множеств, поделенная на область объединения этих множеств (рис. 2.6).

Чем больше пересечений двух множеств, созданных из двух текстов, тем более вероятно, что написаны они одним автором. В листинге 2.8 определяется функция для измерения сходства множества образцов.

Листинг 2.8. Определение функции jaccard_test()

stylometry.py, часть 8

```
def jaccard_test(words_by_author, len_shortest_corpus):
    """Вычислить коэффициент сходства Жаккара каждого корпуса
    к неизвестному корпусу"""
    jaccard_by_author = dict()
    unique_words_unknown = set(words_by_author['unknown']
                               [:len_shortest_corpus])
 ❶ authors = (author for author in words_by_author if author != 'unknown')
    for author in authors:
        unique_words_author = set(words_by_author[author][:len_shortest_
               corpus])
```

```
        shared_words = unique_words_author.intersection(unique_words_unknown)
      ❷ jaccard_sim = (float(len(shared_words))/ (len(unique_words_author) +
                                                  len(unique_words_unknown) -
                                                  len(shared_words)))
        jaccard_by_author[author] = jaccard_sim
        print('Jaccard Similarity for {} = {}'.format(author, jaccard_sim))
  ❸ most_likely_author = max(jaccard_by_author, key=jaccard_by_author.get)
    print('Most-likely author by similarity is {}'.format(most_likely_
                       author))

if __name__ == '__main__':
    main()
```

По аналогии со многими предыдущими тестами функция `jaccard_test()` получает в качестве аргументов словарь слов и длину самого короткого корпуса. Также необходимо, чтобы словарь содержал коэффициент Жаккара для каждого автора.

Рис. 2.6. Пересечение относительно объединения для множества — это область пересечения, поделенная на область объединения

Коэффициент Жаккара работает с уникальными словами, поэтому потребуется преобразовать корпус в множества, чтобы избавиться от повторений. Сначала мы создадим множество из корпуса `unknown`. Далее переберем известные корпусы, преобразуя их во множества и сравнивая со множеством `unknown`. Не забудьте при создании множеств обрезать все корпусы до длины самого короткого.

Прежде чем выполнять цикл, используйте выражение-генератор для получения из словаря `words_by_author` имен авторов за исключением `unknown` ❶. *Выражение-генератор* (generator expression) — это функция, возвращающая объект, значения которого можно перебрать поочередно. Оно во многом похоже на списковое включение, но вместо квадратных скобок заключается в кавычки. При этом выражение-генератор вместо построения требующего значительного объема памяти списка элементов получает элементы в реальном времени. Генераторы полезны при работе с большими наборами значений, которые нужно использовать всего раз. Здесь я применю его, чтобы показать, как он работает.

Когда вы присваиваете выражение-генератор переменной, то получаете только тип итератора под названием *«объект генератора»* (generator object). Сравните этот процесс с созданием списка:

```
>>> mylist = [i for i in range(4)]
>>> mylist
[0, 1, 2, 3]
>>> mygen = (i for i in range(4))
>>> mygen
<generator object <genexpr> at 0x000002717F547390>
```

Выражение-генератор в предыдущем фрагменте полностью аналогично следующей функции-генератору:

```
def generator(my_range):
    for i in range(my_range):
        yield i
```

В то время как инструкция `return` завершает функцию, инструкция `yield` *приостанавливает* ее выполнение и отправляет значение обратно тому, кто ее вызвал. Позже функция сможет возобновить выполнение с момента, на котором остановилась. Когда генератор достигает конца, он оказывается «пуст» и повторно не может быть вызван.

Вернемся к коду. Запускаем цикл `for` при помощи генератора `authors`. Находим уникальные слова для каждого известного автора, как делали это для `unknown`. Затем используем встроенную функцию `intersection()` для нахождения всех слов, общих для множества известного автора и множества `unknown`. Их *пересечение* представляет наибольшее множество, содержащее все элементы,

встречающиеся в них обоих. На основе этой информации можно вычислить коэффициент Жаккара ❷.

Обновляем словарь `jacard_by_author` и выводим каждый результат в окне интерпретатора. Затем находим автора с максимальным значением Жаккара ❸ и выводим результаты.

```
Коэффициент сходства Жаккара для doyle = 0.34847801578354004
Коэффициент сходства Жаккара для wells = 0.30786921307869214
Наиболее вероятным автором по схожести является doyle
```

В итоге счет оказывается в пользу Дойла.

Завершаем stylometry.py кодом для выполнения программы в качестве импортируемого модуля или в автономном режиме.

Итоги

Истинный автор «Затерянного мира» — Дойл. Установив это, мы останавливаемся и объявляем победу. Если вы хотите провести дополнительные исследования, то можете в качестве следующего теста добавить другие известные тексты к `doyle` и `wells`, чтобы их совмещенная длина была ближе к тексту «Затерянного мира» и не пришлось его слишком обрезать. Можно также протестировать длину предложений и стиль пунктуации или применить более сложные техники, такие как нейронные сети и генетические алгоритмы.

Помимо этого, иногда полезно доработать существующие функции, например `vocab_test()` и `jaccard_test()`, техниками *стемминга* и *лемматизации*, чтобы сократить слова до их корневых форм для лучшего сравнения. В том виде, в каком сейчас написана программа, *talk*, *talking* и *talked* рассматриваются как совершенно разные слова, несмотря на общий корень.

И все же стилометрия не может с абсолютной достоверностью гарантировать, что именно сэр Артур Конан Дойл написал «Затерянный мир». Она может лишь предположить на основе весомых свидетельств, что он более вероятный автор, нежели Уэллс. Здесь очень важна формулировка вопроса, поскольку нельзя оценить всех возможных авторов. По этой причине успешное определение авторства начинается со старой доброй детективной работы, которая сокращает список кандидатов до приемлемой длины.

Дополнительная литература

«Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit» (O'Reilly, 2009) Стивена Берда (Steven Bird), Эвана Клейна

(Ewan Klein) и Эдварда Лопера (Edward Loper)[1] представляет доступное введение в область NLP при помощи Python со множеством упражнений и полезной интеграцией с сайтом NLTK. Новая версия книги, обновленная для Python 3 и NLTK 3, доступна онлайн на http:www.nltk.org/book/.

В 1995 году романист Курт Воннегут высказал идею, что «истории имеют формы, которые можно нарисовать на миллиметровой бумаге», и предложил «передать их на обработку компьютерам». В 2018 году энтузиасты реализовали его идею, использовав более 1700 английских романов. Они применили NLP-технику *анализа тональности текста*, которая распознает присущий словам эмоциональный окрас. Полученные результаты описаны в книге «Every Story in the World Has One of These Six Basic Plots», которую можно найти на сайте BBC.com по ссылке http://www.bbc.com/culture/story/20180525-every-story-in-the-world-has-one-of-these-six-basic-plots/.

Практический проект: охота на собаку Баскервилей с помощью распределения

В NLTK есть небольшая занятная функция, называемая *диаграммой распределения*, которая позволяет определить местоположение слова в тексте. Если конкретнее, то она составляет диаграмму вхождений слова относительно количества предшествующих ему слов с начала корпуса.

На рис. 2.7 показана диаграмма распределения упоминания главных персонажей в книге «Собака Баскервилей».

Если вы знакомы с этой историей — а если нет, то спойлерить я ее не стану, — то согласитесь с редким упоминанием Холмса в середине, практически двухполярным распределением вхождений Мортимера и возникшим под конец рассказа пересечением упоминаний о Бэрриморе, Сэлдене и собаке Баскервилей.

Диаграммы распределения имеют и более практическое применение. Например, как автор технической литературы, я должен давать определение новому термину при его первичном появлении в книге. Звучит это просто, но иногда главы пишутся не по порядку, и подобные нюансы проскакивают незамеченными. Диаграмма распределения, построенная для списка технических терминов, если их много, может существенно упростить поиск места их первого упоминания.

[1] Эта книга не издавалась на русском языке, но можно порекомендовать другие издания, имеющие высокий рейтинг: Бенгфорт Б., Ребекка Билбро Р., Тони Охеда Т. «Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка» или Хобсон Л., Ханнес Х., Коул Х. «Обработка естественного языка в действии», обе — издательство «Питер». — *Примеч. ред.*

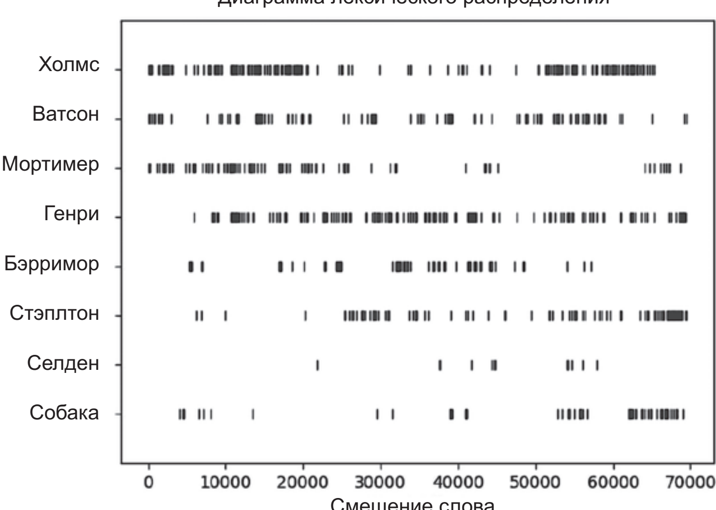


Рис. 2.7. График распределения упоминания главных персонажей в книге «Собака Баскервилей»

Или вот другой вариант. Представьте, что вы — аналитик данных и работаете с ассистентами адвоката по уголовному делу, связанному с торговлей внутренней информацией компании. Для выяснения, говорил ли обвиняемый с конкретным членом правления как раз перед совершением незаконной продажи, вы можете загрузить истребованные электронные письма обвиняемого в виде непрерывной строки и сгенерировать диаграмму распределения. Если имя члена правления появится в ожидаемом месте, то дело можно считать решенным!

Для этого практического проекта напишите на Python программу, воссоздающую диаграмму распределения с рис. 2.7. Если у вас возникнут проблемы с загрузкой корпуса hound.txt, обратитесь еще раз к обсуждению Юникода на с. 66. Решение под названием practice_hound_dispersion.py вы найдете в приложении к книге или онлайн.

Практический проект: тепловая карта пунктуации

Тепловая карта — это диаграмма, в которой значения данных представляются с помощью цвета. Тепловые карты используются для визуализации особенностей авторской пунктуации (https://www.fastcompany.com/3057101/the-surprising-punctuation-habits-of-famous-authors-visualized/) и могут оказаться полезными при установлении авторства «Затерянного мира».

Напишите программу Python, токенизирующую три рассмотренные в этой главе книги на основе только пунктуации. Далее сосредоточьтесь исключительно на использовании точек с запятой. Для каждого автора составьте тепловую карту, отображающую точки с запятыми синим, а все остальные знаки препинания — желтым или красным. На рис. 2.8 показаны примеры тепловых карт для «Войны миров» Уэллса и «Собаки Баскервилей» Дойла.

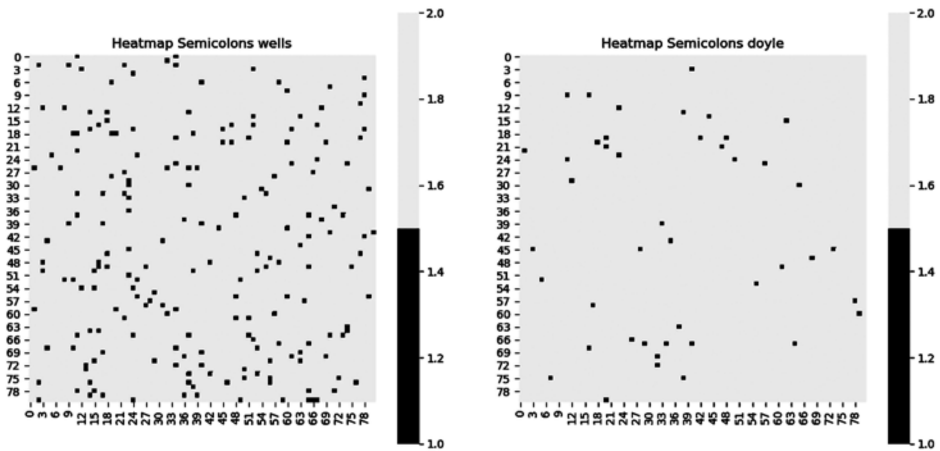


Рис. 2.8. Тепловая карта использования точек с запятой (*темные квадраты*) Уэллсом (*слева*) и Дойлом (*справа*)

Сравните эти три тепловые карты. За чьим авторством, согласно результатам, был написан роман «Затерянный мир» — Дойла или Уэллса?

Решение под названием practice_heatmap_semicolon.py вы найдете в приложении к книге или онлайн.

Усложняем проект: фиксирование частотности

Как уже ранее отмечалось, частотность в NLP означает количество, но также может быть выражена в виде числа вхождений за единицу времени. Еще она выражается как соотношение или процент.

Определите новый вариант метода `nltk.FreqDist()`, который вместо количеств будет использовать проценты, и задействуйте его для формирования графиков в программе stylometry.py. За помощью можете обратиться к блогу Clearly Erroneous на https://martinapugliese.github.io/tech/plotting-the-actual-frequencies-in-a-FreqDist-in-nltk/.

3

Суммаризация текста с помощью обработки естественного языка

«Кругом вода, но не испить ни капли, ни глотка». Известная строчка из поэмы «Сказание о старом мореходе» обобщает текущее положение дел в сфере цифровой информации. Согласно International Data Corporation, к 2025 году мы будем генерировать 175 триллионов гигабайт цифровой информации *в год*. Но большая часть этих данных — до 95 % — будет *неструктурированной*, то есть не организованной в полезные базы данных. Даже сейчас информация о том, как исцелиться от рака, может находиться буквально у нас под носом, но мы об этом не узнаем.

Чтобы упростить поиск и использование информации, необходимо уменьшить объем данных путем экстрагирования и повторной упаковки значимых фрагментов в удобные для восприятия сводки. Из-за огромного объема данных сделать это вручную не представляется возможным. К счастью, обработка естественного языка (NLP) помогает компьютерам понимать как слова, так и контекст. К примеру, приложения NLP способны обобщать новостные ленты, анализировать юридические контракты, исследовать патенты, изучать финансовые рынки, анализировать корпоративные данные и производить руководства к обучению.

В этой главе мы с помощью Natural Language Processing Toolkit (NLTK) сгенерируем суммаризацию одной из известнейших речей всех времен — «I have

a dream» («У меня есть мечта») Мартина Лютера Кинга-младшего. Затем на основе базового понимания используем упрощенную альтернативу, называемую `gensim`, чтобы сделать суммаризацию популярной речи «Make Your Bed» («Заправляй кровать») адмирала Уильяма Макрейвена. В завершение же применим облако слов для создания забавного визуального обобщения слов, наиболее часто используемых в романе Артура Конан Дойла «Собака Баскервилей».

Проект #3. У меня есть мечта... суммаризация речи!

В машинном обучении и дата майнинге есть два подхода к суммаризации текста: *экстракция* и *абстракция*.

При экстрактивной суммаризации используются различные весовые функции, ранжирующие предложения по предполагаемой важности. Слова, используемые чаще, рассматриваются как более важные. Следовательно, предложения с этими словами считаются более важными. Это подобно использованию желтого маркера для пометки вручную ключевых слов и предложений без изменения текста. Результаты могут получаться обрывистыми, но эта техника неплохо справляется с извлечением важных слов и фраз.

Абстракция основана на углубленном понимании документа с целью выделения основных мыслей и его перефразировании более человеческим языком. Это подразумевает создание новых предложений. В данном случае результаты получаются, как правило, более связными и грамматически верными, чем при экстракции, но за это приходится платить. Алгоритмы абстракции требуют использования продвинутых и сложных методов глубокого обучения и комплексного языкового моделирования.

В этом проекте мы будем использовать технику экстракции для суммаризации речи «У меня есть мечта», произнесенной Мартином Лютером Кингом-младшим у мемориала Линкольну 28 августа 1963 года. Как и «Геттисбергское обращение» Линкольна веком ранее, эта идеальная речь была произнесена в идеальное для нее время. Мастерское использование доктором Кингом повторений также делает ее прекрасным образцом для применения техник экстракции, которые соотносят частотность слов с важностью.

ЗАДАЧА

Написать программу Python, которая осуществляет суммаризацию речи при помощи NLP-техники экстракции.

Стратегия

В NLTK есть функции, которые нам понадобятся для суммаризации речи доктора Кинга. Если вы пропустили главу 2, то вернитесь на с. 59, где описаны инструкции по установке этого набора инструментов.

Для суммаризации речи вам понадобится ее текст в электронном виде. В предыдущих главах вы самостоятельно скачивали нужные файлы из интернета.

На этот раз мы воспользуемся более эффективной техникой, называемой *веб-скрапингом*, которая позволяет программно извлекать и сохранять большие объемы данных с веб-сайтов.

После скачивания речи в виде строки мы сможем использовать NLTK для ее разделения на отдельные слова и их подсчета. Затем присвоим каждому предложению речи «балл» путем суммирования содержащихся в нем слов. Эти баллы мы используем для вывода самых значимых предложений, ориентируясь на нужный объем итогового резюме.

Веб-скрапинг

Веб-скрапинг означает использование программы для скачивания и обработки контента. Это настолько распространенная задача, что бесплатно предлагается немало предназначенных для этого приложений. Мы с вами воспользуемся библиотекой `requests` для скачивания файлов и веб-страниц, а также пакетом Beautiful Soup (`bs4`) для парсинга HTML. HTML (*Hypertext Markup Language*) — это *язык гипертекстовой разметки*, стандартный формат для создания веб-страниц.

Для установки этих двух модулей используйте pip в окне терминала или Windows PowerShell (инструкции по установке и использованию pip содержатся на с. 33 главы 1):

```
pip install requests
pip install beautifulsoup4
```

Для проверки, что установка прошла успешно, откройте оболочку и импортируйте каждый модуль, как показано ниже. Если ошибки не возникнет, значит, все хорошо.

```
>>> import requests
>>>
>>> import bs4
>>>
```

За более подробной информацией о `requests` обратитесь на https://pypi.org/project/requests/. Подробнее о Beautiful Soup можно узнать на https://www.crummy.com/software/BeautifulSoup/.

Код для «У меня есть мечта»

Программа dream_summary.py выполняет следующие действия.

1. Открывает веб-страницу, содержащую речь «У меня есть мечта».
2. Скачивает ее текст в виде строки.
3. Токенизирует текст на слова и предложения.
4. Удаляет стоп-слова, не влияющие на контекст.
5. Подсчитывает оставшиеся слова.
6. Использует полученные количества слов для ранжирования предложений.
7. Отображает самые значимые предложения.

Если вы уже скачали файлы книги, то найдите программу в каталоге Chapter_3. В противном случае перейдите на https://nostarch.com/real-world-python/ и скачайте ее с GitHub.

Импорт модулей и определение функции main()

Листинг 3.1 импортирует модули и определяет первую часть функции `main()`, которая делает скрапинг и присваивает текст речи переменной в качестве строки.

Листинг 3.1. Импорт модулей и определение функции main()

dream_summary.py, часть 1

```
from collections import Counter
import re
import requests
import bs4
import nltk
from nltk.corpus import stopwords

def main():
❶   url = 'http://www.analytictech.com/mb021/mlk.htm'
    page = requests.get(url)
    page.raise_for_status()
❷   soup = bs4.BeautifulSoup(page.text, 'html.parser')
    p_elems = [element.text for element in soup.find_all('p')]

    speech = ''.join(p_elems)
```

Начинаем с импорта `Counter` из модуля `collections`, который пригодится для отслеживания баллов предложений. Модуль `collections` является частью Python Standard Library и включает несколько контейнерных типов данных. `Counter` — это подкласс словаря для подсчета хешируемых объектов. Элементы сохраняются как ключи словаря, а их количества — как значения.

Далее для очистки речи перед суммаризацией ее содержимого импортируем модуль `re`. Сокращение `re` означает *регулярные выражения (regular expressions, regex)*, которые представляют собой последовательности символов, определяющих шаблон поиска. Этот модуль поможет очистить речь, позволив выборочно удалять ненужные части.

Завершаем импорт модулями для скрапинга и обработки естественного языка. Последний модуль предоставляет список функциональных стоп-слов (таких, как *if*, *and*, *but*, *for*), которые не содержат полезной информации. Эти слова мы удалим из речи до ее суммаризации.

Далее следует определить функцию `main()` для запуска программы. Чтобы сделать скрапинг речи из интернета, нужно предоставить `url` адрес в качестве строки ❶. Для этого можно скопировать его с сайта, откуда требуется извлечь текст.

Библиотека `requests` абстрагирует сложности создания HTTP-запроса в Python. HTTP (HyperText Transfer Protocol) — протокол передачи гипертекста — лежит в основе передачи данных по всемирной сети при помощи гиперссылок. С помощью метода `requests.get()` получаем `url` и присваиваем вывод переменной `page`, которая ссылается на объект `Response`, возвращенный на запрос веб-страницей. Текстовый атрибут этого объекта содержит веб-страницу, включая текст, в виде строки.

Чтобы убедиться в успешности скачивания, вызываем метод `raise_for_status()` объекта `Response`. Если все в порядке, то при вызове ничего не происходит, в противном же случае будет выдано исключение и программа остановится.

В этот момент данные находятся в формате HTML, как показано ниже:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<title>Martin Luther King Jr.'s 1962 Speech</title>
</head>
--snip--
<p>I am happy to join with you today in what will go down in
history as the greatest demonstration for freedom in the history
of our nation. </p>
--snip--
```

Как видите, в HTML множество *тегов*, таких как `<head>` и `<p>`, которые позволяют браузеру понять, как форматировать веб-страницу. Текст между открывающим и закрывающим тегами называется *элементом*. К примеру, текст «Martin Luther King Jr.'s 1962 Speech» — это элемент заголовка, вставленный между открывающим тегом `<title>` и закрывающим `</title>`. Абзацы форматируются при помощи тегов `<p>` и `</p>`.

Так как эти теги не являются частью оригинального текста, то, прежде чем приступать к обработке естественного языка, их нужно удалить. Для этого вызываем метод `bs4.BeautifulSoup()` и передаем ему строку, содержащую HTML ❷. Заметьте, что я явно указал `html.parser`. Программа работает и без него, но при этом она будет выдавать предупреждения в оболочке.

Теперь переменная `soup` ссылается на объект `BeautifulSoup`, то есть далее можно использовать метод объекта `find_all()` для обнаружения речи, содержащейся в HTML-документе. В этом случае, чтобы найти текст между тегами абзацев `<p>`, используется списковое включение и `find_all()`, создающее список, содержащий только элементы абзацев.

В заключение преобразуем речь в непрерывную строку. С помощью метода `join()` превращаем список `p.elems` в строку. Устанавливаем символ «объединитель» как пробел, ограниченный кавычками ' '.

Обратите внимание, что при использовании Python обычно существует несколько способов решения задачи. Последние две строки листинга можно написать и так:

```
p_elems = soup.select('p')
speech = ' '.join(p_elems)
```

Метод `select()` в целом более ограничен, нежели `find_all()`, но в этом случае он работает точно так же и оказывается более кратким. В предыдущем сниппете `select()` находит теги `<p>`, и результаты преобразуются в текст, конкатенированный в строку `speech`.

Завершение функции main()

Далее мы подготовим речь для исправления опечаток и удаления пунктуации, специальных символов, а также пробелов. После этого вызовем три функции для удаления стоп-слов, подсчета частотности слов и оценки предложений на основе количества слов. В завершение мы ранжируем предложения и отобразим самые значимые из них в оболочке.

В листинге 3.2 завершается определение функции `main()`, выполняющей эти задачи.

Листинг 3.2. Завершение функции main()

dream_summary.py, **часть 2**

```
    speech = speech.replace(')mowing', 'knowing')
    speech = re.sub('\s+', ' ', speech)
    speech_edit = re.sub('[^a-zA-Z]', ' ', speech)
    speech_edit = re.sub('\s+', ' ', speech_edit)

❶ while True:
        max_words = input("Enter max words per sentence for summary: ")
        num_sents = input("Enter number of sentences for summary: ")
        if max_words.isdigit() and num_sents.isdigit():
            break
        else:
            print("\nInput must be in whole numbers.\n")

    speech_edit_no_stop = remove_stop_words(speech_edit)
    word_freq = get_word_freq(speech_edit_no_stop)
    sent_scores = score_sentences(speech, word_freq, max_words)

❷ counts = Counter(sent_scores)
    summary = counts.most_common(int(num_sents))
    print("\nSUMMARY:")
    for i in summary:
        print(i[0])
```

Оригинальный документ содержит опечатку (*mowing* вместо *knowing*), поэтому начнем с ее исправления при помощи метода `string.replace()`. Продолжаем чистить текст, используя `regex`. Многих программистов отталкивает магический синтаксис этого модуля, но `regex` настолько эффективен и полезен, что знать его базовый синтаксис необходимо каждому.

Удаляем лишние пробелы с помощью функции `re.sub()`, которая заменяет подстроки новыми символами. Применяем сокращенный код символьного класса `\s+`, чтобы определить повторяющиеся пробелы и заменить их одним пробелом, который обозначается `' '`. Завершаем, передавая в `re.sub()` имя строки (`speech`).

Теперь удаляем все *не* буквы путем сопоставления с шаблоном `[^a-zA-Z]`. Значок `^` инструктирует `regex` «сопоставлять все символы, не находящиеся в скобках». Таким образом, числа, знаки препинания и т. п. будут заменены пробелом.

После удаления таких символов, как знаки препинания, останутся лишние пробелы. Чтобы от них избавиться, еще раз вызываем метод `re.sub()`.

Далее вставляем запрос к пользователю на ввод количества включаемых в обобщение предложений и максимального количества слов в предложении. С помощью цикла `while` и встроенной функции Python `isdigit()` убеждаемся в том, что пользователь вводит целые числа ❶.

ПРИМЕЧАНИЕ

Согласно исследованию, проведенному Американским институтом прессы, смысл текста или речи лучше всего передается при помощи предложений размером до 15 слов. В Оксфордском руководстве по разговорному английскому языку также рекомендуется использовать предложения, в среднем содержащие от 15 до 20 слов.

Продолжаем чистку текста, вызывая функцию `remove_stop_words()`. Далее вызываем функции `set_word_freq()` и `score_sentences()` для вычисления частотности оставшихся слов и оценки предложений соответственно. Эти функции мы определим после завершения функции `main()`.

Для ранжирования предложений вызываем метод `Counter()` из модуля `Collection` ❷, которому передаем переменную `sent_scores`.

Для генерации резюме используем метод `most_common()` объекта `Counter`. В него передаем переменный ввод пользователя `num_sents`. Переменная `summary` будет содержать список кортежей. В каждом кортеже предложение занимает индекс `[0]`, а его ранг — индекс `[1]`.

```
[('From every mountainside, let freedom ring.', 4.625), --snip-- ]
```

Для лучшей читаемости выводим каждое предложение резюме на отдельной строке.

Удаление стоп-слов

Как я говорил в главе 2, стоп-слова — это короткие функциональные слова вроде *if*, *but*, *for* и *so*. Так как они не содержат никакой важной контекстной информации, использовать их для ранжирования предложений не нужно.

В листинге 3.3 определяется функция `remove_stop_words()`, удаляющая из речи стоп-слова.

Листинг 3.3. Определение функции для удаления из речи стоп-слов

dream_summary.py, часть 3

```
def remove_stop_words(speech_edit):
    """Удалить из строки стоп-слова и вернуть строку."""
    stop_words = set(stopwords.words('english'))
    speech_edit_no_stop = ''
    for word in nltk.word_tokenize(speech_edit):
        if word.lower() not in stop_words:
            speech_edit_no_stop += word + ' '
    return speech_edit_no_stop
```

Определяем функцию для получения в качестве аргумента отредактированной строки речи `speech_edit`. Далее создаем множество английских стоп-слов

в NLTK. Используем именно множество, а не список, так как по нему поиск выполняется быстрее.

Назначаем пустую строку для хранения отредактированной речи без стоп-слов. Сейчас переменная `speech_edit` является строкой, в которой каждый элемент — это буква.

Для работы со словами вызываем NLTK-метод `word_tokenizer()`. Обратите внимание, что можно сделать это, перебирая слова. Преобразуем каждое слово в нижний регистр и проверяем, находится ли оно во множестве `stop_words`. Если это не стоп-слово, конкатенируем его с новой строкой вместе с пробелом. В завершение возвращаем функции эту строку.

В этой программе важен способ обработки регистра букв. Нам нужно вывести резюме с буквами как верхнего, так и нижнего регистра, но, чтобы избежать ошибок в подсчетах при NLP-обработке, все буквы должны быть в нижнем регистре. Чтобы понять почему, взгляните на следующий фрагмент кода, в котором подсчитываются слова в строке (`s`) со смешанными регистрами:

```
>>> import nltk
>>> s = 'one One one'
>>> fd = nltk.FreqDist(nltk.word_tokenize(s))
>>> fd
FreqDist({'one': 2, 'One': 1})
>>> fd_lower = nltk.FreqDist(nltk.word_tokenize(s.lower()))
>>> fd_lower
FreqDist({'one': 3})
```

Если не перевести слова в нижний регистр, *one* и *One* будут считаться по отдельности. Для подсчета каждый экземпляр *one* независимо от регистра должен рассматриваться как одно слово. В противном случае участие *one* в документе будет недооценено.

Вычисление частотности вхождений слов

Для подсчета вхождений каждого слова в речи мы создадим функцию `get_word_freq()`, возвращающую словарь со словами в качестве ключей и их количеством в качестве значений. Эта функция определяется в листинге 3.4.

Листинг 3.4. Определение функции для вычисления частотности слов в речи

dream_summary.py, часть 4

```
def get_word_freq(speech_edit_no_stop):
    """Возвращаем словарь с частотностью слов в строке"""
    word_freq = nltk.FreqDist(nltk.word_tokenize(speech_edit_no_stop.lower()))
    return word_freq
```

Функция `get_words_freq()` получает в качестве аргумента отредактированную строку речи без стоп-слов. Класс NLTK `FreqDist` выступает как словарь со словами в качестве ключей и их количеством в качестве значений. Таким образом мы преобразуем входную строку в нижний регистр и токенизируем ее на слова. Завершим функцию возвращением словаря `word_freq`.

Оценка предложений

Функция в листинге 3.5 оценивает предложения на основе распределения частотности слов в этих предложениях. Она возвращает словарь, где предложение выступает ключом, а его балл (score) — значением.

Листинг 3.5. Определение функции для оценки предложений на основе частотности слов в них

dream_summary.py, часть 5

```
def score_sentences(speech, word_freq, max_words):
    """Возвращаем словарь оценок предложений на основе частотности слов"""
    sent_scores = dict()
    sentences = nltk.sent_tokenize(speech)
  ❶ for sent in sentences:
        sent_scores[sent] = 0
        words = nltk.word_tokenize(sent.lower())
        sent_word_count = len(words)
      ❷ if sent_word_count <= int(max_words):
            for word in words:
                if word in word_freq.keys():
                    sent_scores[sent] += word_freq[word]
          ❸ sent_scores[sent] = sent_scores[sent] / sent_word_count
    return sent_scores

if __name__ == '__main__':
    main()
```

Определяем функцию `score_sentences()` с параметрами для оригинальной строки `speech`, объекта `word_freq` и переменного пользовательского ввода `max_words`. Нам нужно, чтобы резюме также содержало стоп-слова и слова с заглавных букв, поэтому и используется `speech`.

Создаем пустой словарь `sent_scores`, который будет хранить баллы для каждого предложения. Далее токенизируем строку `speech` на предложения.

Выполняем перебор предложения ❶. Начинаем с обновления словаря `sent_scores`, где устанавливаем предложения в качестве ключей, а их начальные значения (количества) как `0`.

Для подсчета частотности слов сначала нужно токенизировать предложения на слова. Не забудьте использовать нижний регистр для совместимости со словарем `word_freq`.

Необходимо внимание при суммировании количеств вхождений слов каждого предложения для подсчета его баллов, чтобы не исказить результаты в пользу более длинных предложений. Ведь чем длиннее предложение, тем выше вероятность встретить в нем важные слова. Чтобы избежать исключения коротких, но важных предложений, нужно *нормализовать* каждый подсчет слов, разделив его на *длину* предложения. Длину мы сохраняем в переменной `sent_word_count`.

Далее используем условие, которое ограничивает предложения максимальной заданной пользователем длиной ❷. Если предложение удовлетворяет условию, начинаем перебирать его слова. Если слово оказывается в словаре `word_freq`, добавляем его к количеству, содержащемуся в `sent_scores`.

В конце каждого цикла перебираем предложения и делим балл текущего предложения на количество слов в нем ❸. Так мы нормализуем баллы длинных предложений, чтобы они не получили несправедливого преимущества.

Завершается функция возвращением словаря `sent_scores`. Далее, снова в глобальной области, добавляем код для выполнения программы в качестве модуля или в автономном режиме.

Выполнение программы

Выполним программу dream_summary.py, задав максимальное количество слов в предложениях равным 14. Как уже говорилось, хорошие, читаемые предложения содержат до 14 слов. Далее ограничиваем резюме до 15 предложений, что представляет примерно одну третью часть текста. Ниже показан результат, который у вас должен получиться. Обратите внимание, что предложения не обязательно расположены в изначальном порядке.

```
Ввести максимальное количество слов в предложении для суммаризации: 14
Ввести количество предложений в резюме: 15

SUMMARY:
From every mountainside, let freedom ring.
Let freedom ring from Lookout Mountain in Tennessee!
Let freedom ring from every hill and molehill in Mississippi.
Let freedom ring from the curvaceous slopes of California!
Let freedom ring from the snow capped Rockies of Colorado!
But one hundred years later the Negro is still not free.
From the mighty mountains of New York, let freedom ring.
From the prodigious hilltops of New Hampshire, let freedom ring.
And I say to you today my friends, let freedom ring.
I have a dream today.
It is a dream deeply rooted in the American dream.
Free at last!
Thank God almighty, we're free at last!"
We must not allow our creative protest to degenerate into physical violence.
This is the faith that I go back to the mount with.
```

Суммаризация не только учитывает заголовок речи, но также включает ее основные моменты.

Если же выполнить программу еще раз, но ограничить предложения десятью словами, то многие предложения окажутся слишком длинными. Поскольку во всей речи содержится всего 7 предложений с количеством слов до 10, программа не может удовлетворить входным требованиям. Она по умолчанию выводит речь с начала и до момента, пока количество предложений не достигнет заданного в переменной num_sents значения.

Теперь еще раз выполним программу, ограничив длину предложений 1000 словами.

```
Ввести максимальное количество слов в предложении для суммаризации: 1000
Ввести количество предложений в резюме: 15

SUMMARY:
From every mountainside, let freedom ring.
Let freedom ring from Lookout Mountain in Tennessee!
Let freedom ring from every hill and molehill in Mississippi.
Let freedom ring from the curvaceous slopes of California!
Let freedom ring from the snow capped Rockies of Colorado!
But one hundred years later the Negro is still not free.
From the mighty mountains of New York, let freedom ring.
From the prodigious hilltops of New Hampshire, let freedom ring.
And I say to you today my friends, let freedom ring.
I have a dream today.
But not only there; let freedom ring from the Stone Mountain of Georgia!
It is a dream deeply rooted in the American dream.
With this faith we will be able to work together, pray together; to struggle
together, to go to jail together, to stand up for freedom forever, knowing
that we will be free one day.
Free at last!
One hundred years later the life of the Negro is still sadly crippled by the
manacles of segregation and the chains of discrimination.
```

Хотя более длинные предложения и не доминируют в резюме, несколько таких все же сюда попали, сделав его не столь поэтичным, как предыдущее. Ограничение предложений меньшим количеством слов вынуждает предыдущую версию полагаться на более короткие фразы, которые выступают в качестве своеобразного рефрена.

Проект #4. Суммаризация речи с помощью gensim

В эпизоде «Симпсонов», удостоенном, кстати, награды «Эмми», Гомер баллотируется в комиссию по санитарии под лозунгом «Не может ли это сделать кто-то

другой?». То же и со многими приложениями Python: зачастую, когда нужно написать скрипт, вы узнаете, что это уже кто-то сделал. Один из примеров — `gensim`, открытая библиотека для обработки естественного языка при помощи статистического машинного обучения.

Слово *gensim* означает «генерация подобного» (generate similar). В этой библиотеке используется основанный на графах алгоритм ранжирования TextRank. Создан он был на базе PageRank, разработанного Ларри Пейджем (Larry Page) и применяемого для ранжирования веб-страниц в поиске Google. В случае PageRank важность веб-сайта определяется количеством ссылок на него с других страниц. Чтобы использовать этот подход в обработке текста, алгоритмы измеряют, насколько каждое предложение похоже на другие. Предложение, имеющее наибольшее сходство, считается наиболее важным.

В текущем проекте мы задействуем `gensim` для обобщения напутственной речи адмирала Уильяма Макрейвена «Заправляйте свою кровать» («Make Your Bed»), с которой он выступил в Университете Техаса в Остине в 2014 году. Эта мотивирующая 20-минутная речь собрала более 10 миллионов просмотров на YouTube; позже на ее основе была издана книга, ставшая в 2017 году бестселлером *New York Times*.

ЗАДАЧА

Написать программу Python, использующую модуль gensim для суммаризации речи.

Установка gensim

Модуль `gensim` работает на всех ведущих операционных системах, но зависит от `NumPy` и `SciPy`. Если эти библиотеки у вас не установлены, вернитесь к главе 1 и выполните инструкции из раздела «Установка библиотек Python» на с. 31.

Чтобы установить `gensim` в Windows, используйте `pip install -U gensim`. Для установки из терминала используйте `pip install --upgrade gensim`. Для окружения `conda` используйте `conda -c conda-forge gensim`. Подробнее о `gensim` вы можете узнать на странице https://radimrehurek.com/gensim/.

Код для суммаризации речи «Заправляйте свою кровать»

Проработав программу dream_summary.py в проекте 3, вы познакомились с основами извлечения текста. Поскольку кое-какие детали вы уже знаете, то можете использовать `gensim` в качестве упрощенной альтернативы dream_summary.py. Назовите эту новую программу bed_summary.py или скачайте ее с сайта книги.

Импорт модулей, веб-скрапинг и подготовка строки речи

В листинге 3.6 повторяется код из dream_summary.py, использованный для подготовки речи в качестве строки. Описание этого кода находится на с. 88.

Листинг 3.6. Импорт модулей и скачивание речи как строки

bed_summary.py, часть 1

```
   import requests
   import bs4
   from nltk.tokenize import sent_tokenize
❶  from gensim.summarization import summarize

❷  url = 'https://jamesclear.com/great-speeches/make-your-bed-by-admiral
          -william-h-mcraven'
   page = requests.get(url)
   page.raise_for_status()
   soup = bs4.BeautifulSoup(page.text, 'html.parser')
   p_elems = [element.text for element in soup.find_all('p')]

   speech = ' '.join(p_elems)
```

Мы будем тестировать `gensim` на необработанной речи, взятой из интернета, поэтому модули для ее очистки вам не понадобятся. Модуль `gensim` производит все подсчеты внутренне, то есть `Counter` не понадобится, но потребуется `gensim`-функция `summarize()`, которая будет делать суммаризацию текста ❶. Еще одно отличие — это адрес `url` ❷.

Суммаризация речи

Листинг 3.7 завершает программу, делает суммаризацию речи и выводит результаты.

Листинг 3.7. Выполнение `gensim`, удаление повторяющихся строк и вывод резюме

bed_summary.py, часть 2

```
print("\nSummary of Make Your Bed speech:")
summary = summarize(speech, word_count=225)
sentences = sent_tokenize(summary)
sents = set(sentences)
print(' '.join(sents))
```

Начинаем с вывода заголовка резюме. Далее вызываем функцию `summarize()`, которая делает суммаризацию речи в 225 слов. Из этого количества слов получится примерно 15 предложений с учетом того, что средняя длина предложения составит 15 слов. Помимо подсчета слов можно передать в `summarize()`

отношение, например `ratio=0.01`. В этом случае длина обобщения составит 1 % от длины документа.

В идеале можно сделать суммаризацию речи и вывести результат за один шаг.

```
print(summarize(speech, word_count=225))
```

К сожалению, иногда `gensim` повторяет предложения в резюме, что мы наблюдаем в следующем примере:

```
Summary of Make Your Bed speech:
Basic SEAL training is six months of long torturous runs in the soft sand,
midnight swims in the cold water off San Diego, obstacle courses, unending
calisthenics, days without sleep and always being cold, wet and miserable.
Basic SEAL training is six months of long torturous runs in the soft sand,
midnight swims in the cold water off San Diego, obstacle courses, unending
calisthenics, days without sleep and always being cold, wet and miserable.
--snip--
```

Чтобы избежать повторов текста, сначала нужно разделить предложения в переменной суммаризации с помощью функции NLTK `sent_tokenize()`. Далее создадим из этих предложений множество, в котором будут удалены повторы. В завершение выведем результат.

Поскольку множества не упорядочены, то при повторном выполнении программы последовательность предложений может изменяться.

```
Summary of Make Your Bed speech:
If you can't do the little things right, you will never do the big things
right.And, if by chance you have a miserable day, you will come home to a
bed that is made — that you made — and a made bed gives you encouragement
that tomorrow will be better.If you want to change the world, start off
by making your bed.During SEAL training the students are broken down into
boat crews. It's just the way life is sometimes.If you want to change the
world get over being a sugar cookie and keep moving forward.Every day during
training you were challenged with multiple physical events — long runs, long
swims, obstacle courses, hours of calisthenics — something designed to test
your mettle. Basic SEAL training is six months of long torturous runs in the
soft sand, midnight swims in the cold water off San Diego, obstacle courses,
unending calisthenics, days without sleep and always being cold, wet and
miserable.
>>>
======= RESTART: C:\Python372\sequel\wordcloud\bed_summary.py =======

Summary of Make Your Bed speech:
It's just the way life is sometimes.If you want to change the world get over
being a sugar cookie and keep moving forward.Every day during training you
were challenged with multiple physical events — long runs, long swims,
obstacle courses, hours of calisthenics — something designed to test your
mettle. If you can't do the little things right, you will never do the big
```

```
things right.And, if by chance you have a miserable day, you will come home to
a bed that is made — that you made — and a made bed gives you encouragement
that tomorrow will be better.If you want to change the world, start off by
making your bed.During SEAL training the students are broken down into boat
crews. Basic SEAL training is six months of long torturous runs in the soft
sand, midnight swims in the cold water off San Diego, obstacle courses,
unending calisthenics, days without sleep and always being cold, wet and
miserable.
```

Если вы прочтете всю речь, то наверняка решите, что `gensim` создает прекрасную суммаризацию. Несмотря на то что эти два результата отличаются, они оба вобрали основные моменты речи, включая фрагмент о заправке кровати. Учитывая размер документа, я нахожу этот результат впечатляющим.

Далее мы рассмотрим другой способ суммаризации текста при помощи слов и облака слов.

Проект #5. Суммаризация речи с помощью облака слов

Облако слов — это визуальное представление текстовых данных, используемое для отображения метаданных ключевых слов; на веб-сайтах их называют *тегами*. В облаке слов размер или цвет шрифта показывают важность каждого тега или слова.

Облако слов позволяет выделять ключевые слова в документе. К примеру, генерация облака слов для каждого доклада президента США конгрессу о положении дел в стране может быстро предоставить обзор проблем, с которыми столкнулось государство в текущем году. В первый год президентства Билл Клинтон делал акцент на проблемах мирного времени, таких как здравоохранение, трудовая занятость населения и налоги (рис. 3.1).

Рис. 3.1. Облако слов, созданное из доклада Билла Клинтона Конгрессу США в 1993 году

Менее чем через 10 лет облако слов Джорджа Буша демонстрирует уже приоритеты в области государственной безопасности (рис. 3.2).

Рис. 3.2. Облако слов, созданное из доклада Джорджа Буша Конгрессу США в 2002 году

Еще одно применение облака слов — извлечение ключевых слов из отзывов покупателей. Если обнаружится преобладание таких слов, как *плохой*, *медленный* и *дорогой*, то это указывает на проблему. Писатели могут также использовать облако для сравнения глав книги или сцен в киносценарии. Если автор использует схожую лексику для экшн-сцен и романтических интерлюдий, то потребуется редактирование. Копирайтеры могут с помощью облака проверять плотность ключевых слов для оптимизации в поисковых системах (SEO).

Генерировать облако слов можно многими способами, включая использование бесплатных сайтов вроде https://www.wordclouds.com/ и https://www.jasondavies.com/wordcloud/. Но если вы хотите полноценно настроить свое облако слов или встроить его генератор в другую программу, то это следует делать самостоятельно. В данном проекте мы используем облако слов, чтобы создать рекламный флаер для школьного спектакля по повести «Собака Баскервилей».

Вместо использования базового прямоугольника, показанного на рис. 3.1 и 3.2, мы будем вписывать слова в контур головы Шерлока Холмса (рис. 3.3).

Рис. 3.3. Силуэт Шерлока Холмса

Так у нас получится более узнаваемое и привлекательное представление.

ЗАДАЧА

Использовать модуль `wordcloud` для генерации фигурного облака слов из повести о Шерлоке Холмсе.

Модули Word Cloud и PIL

Для генерации облака слов мы используем модуль `wordcloud`. Установите его с помощью pip.

```
pip install wordcloud
```

В случае использования Anaconda введите команду:

```
conda install -c conda-forge wordcloud
```

Страница модуля `wordcloud` находится на http://amueller.github.io/word_cloud/.

Вам также понадобится Python Imaging Library (PIL) для работы с изображениями. Для ее установки также используйте pip.

```
pip install pillow
```

А в случае с Anaconda введите команду:

```
conda install -c anaconda pillow
```

К слову, `pillow` является проектом-преемником PIL, поддержка которого была прекращена в 2011 году. Более подробно о нем можете узнать на https://pillow.readthedocs.io/en/stable/.

Код для создания облака слов

Для создания фигурного облака слов потребуются файл изображения и текстовый файл. Изображение, показанное на рис. 3.3, взято с площадки iStock by Getty Images (https://www.istockphoto.com/vector/detective-hat-gm698950970-129478957/). Разрешение этой картинки составляет всего 500×600 пикселей.

Схожее изображение, но уже без авторских прав (holmes.png) содержится среди доступных для скачивания файлов книги. Текстовый файл (hound.txt), файл изображения (holmes.png) и код (wc_hound.py) находятся в каталоге Chapter_3.

Импорт модулей, текстовых файлов, файлов изображений и стоп-слов

В листинге 3.8 мы импортируем модули, скачаем роман и изображение силуэта Холмса, а также создадим множество стоп-слов, которые нужно исключить из облака.

Листинг 3.8. Импорт модулей и скачивание текста, изображения, а также стоп-слов

wc_hound.py, часть 1

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS

# Загрузка текстового файла как строки.
❶ with open('hound.txt') as infile:
    text = infile.read()

# Скачивание изображения в виде массива NumPy.
mask = np.array(Image.open('holmes.png'))

# Получение стоп-слов в виде множества и добавление дополнительных слов.
stopwords = STOPWORDS
❷ stopwords.update(['us', 'one', 'will', 'said', 'now', 'well', 'man', 'may',
                  'little', 'say', 'must', 'way', 'long', 'yet', 'mean',
                  'put', 'seem', 'asked', 'made', 'half', 'much',
                  'certainly', 'might', 'came'])
```

Начинаем с импорта `NumPy` и PIL. PIL открывает изображение, а `NumPy` преобразовывает его в маску. Использовать `NumPy` мы начали в главе 1. Если же вы эту главу пропустили, то обратитесь к разделу «Установка библиотек Python» на с. 31. Имейте в виду, что модуль `pillow` продолжает использовать акроним PIL для обратной совместимости.

Для отображения облака слов понадобится `matplotlib`, которую вы скачали в том же разделе «Установка библиотек Python» главы 1. Модуль `wordcloud` содержит собственный список стоп-слов, поэтому импортируйте `STOPWORDS` вместе с функциональностью облака.

Далее скачайте текстовый файл романа и сохраните его в переменной `text` ❶. Как я говорил, описывая листинг 2.2 главы 2, при скачивании текста вы можете столкнуться с ошибкой `UnicodeDecodeError`.

```
UnicodeDecodeError: 'ascii' codec can't decode byte 0x93 in position 365:
ordinal not in range(128)
```

В этом случае попробуйте изменить функцию `open()`, добавив аргументы `encoding` и `errors`.

```
with open('hound.txt', encoding='utf-8', errors='ignore') as infile:
```

После скачивания текста примените метод PIL `Image.open()`, чтобы открыть изображение Холмса, и с помощью `NumPy` преобразуйте его в массив. Если вы используете изображение с iStock, то измените соответствующим образом название файла.

Далее присваиваем множество `STOPWORDS`, импортированное с `wordcloud`, переменной `stopwords`. Затем обновляем это множество списком дополнительных слов, которые нужно исключить ❷. К ним относятся такие слова, как *said* и *now*, которые доминируют в облаке, но полезной информации не несут. Определение подобных слов происходит итеративно. Генерируется облако слов, из которого удаляются слова, не несущие смысловой нагрузки, после чего процесс повторяется. Если хотите увидеть конкретное влияние этой строки, можете ее закомментировать.

ПРИМЕЧАНИЕ

Чтобы обновить контейнер вроде STOPWORDS, нужно знать, является ли он списком, словарем, множеством или чем-то другим. Встроенная в Python функция type() возвращает тип класса любого объекта, переданного ей в качестве аргумента. В этом случае команда print(type(STOPWORDS)) дает <class 'set'>.

Генерация облака слов

Код листинга 3.9 генерирует облако слов и задействует силуэт в качестве *маски*, иначе говоря, изображения, используемого для сокрытия другого изображения. Процесс, применяемый `wordcloud`, достаточно продуман, чтобы вписать слова в маску, а не просто обрезать их по краям. Кроме того, для изменения представления слов внутри маски также доступно множество параметров.

Листинг 3.9. Генерация облака слов

wc_hound.py, часть 2

```
wc = WordCloud(max_words=500,
               relative_scaling=0.5,
               mask=mask,
               background_color='white',
               stopwords=stopwords,
               margin=2,
               random_state=7,
               contour_width=2,
               contour_color='brown',
               colormap='copper').generate(text)

colors = wc.to_array()
```

Называем переменную `wc` и вызываем `WordCloud()`. Здесь множество параметров, так что я поместил каждый на отдельную строку для наглядности. Список и описание всех доступных параметров вы найдете на https://amueller.github.io/word_cloud/generated/wordcloud.WordCloud.html.

Начинаем с передачи максимального количества слов, которое хотим использовать. Заданная вами величина будет означать *n* наиболее распространенных

слов в тексте. Чем больше слов выбрано для показа, тем проще определить края маски и сделать ее узнаваемой. К сожалению, ввод слишком большого значения для максимума приводит к появлению большого числа мелких, неразборчивых слов. Для данного проекта начнем со значения 500.

Далее для управления размером шрифта и относительной важностью каждого слова устанавливаем для параметра `relative_scaling` значение `0.5`. К примеру, при значении 0 предпочтение при выборе размера шрифта отдается рангу слова, а при значении 1 слова, встречающиеся вдвое чаще, будут иметь вдвое больший размер. Значения между 0 и 0.5 определяют оптимальный баланс между рангом и частотностью.

Ссылаемся на переменную `mask` и устанавливаем ее фоновый цвет как `white`. Если цвет не присвоить, то он по умолчанию — черный. Далее ссылаемся на множество `stopwords`, которое мы редактировали в предыдущем листинге.

Параметр `margin` регулирует интервалы между отображаемыми словами. При значении `0` слова будут расположены очень плотно. При значении `2` между ними уже будет некоторое расстояние.

Для размещения слов по облаку мы используем генератор случайных чисел и устанавливаем значение `random_state` как `7`. В этом значении нет ничего особенного. Просто мне показалось, что при его использовании слова располагаются наиболее привлекательным образом.

Параметр `random_state` фиксирует стартовое число, чтобы результаты удалось повторить при условии сохранения других параметров. Это означает, что слова всегда будут упорядочены одинаково. Здесь можно установить только целочисленное значение.

Теперь задаем для `contour_width` значение `2`. Любое значение больше нуля создает вокруг маски контур. В данном случае контур волнистый, что обусловлено разрешением изображения (рис. 3.4).

С помощью параметра `contour_color` устанавливаем цвет контура как `brown`. Продолжаем использовать оттенки коричневого, установив `colormap` как `copper`. В `matplotlib` *colormap* — это словарь, отображающий числа в цвета. Цветовая карта `copper` создает текст с окраской в диапазоне от бледно-телесного до черного. Его спектр, как и другие опции цвета, можно посмотреть на странице https://matplotlib.org/stable/gallery/color/colormap_reference.html. Если цветовую карту не задать, то программа будет использовать предустановленные цвета.

С помощью точечной нотации вызываем метод `generate()` для построения облака слов. Передаем ему в качестве аргумента строку `text`. Завершается листинг именованием переменной `colors` и вызовом метода `to_array()` для объекта `wc`.

Этот метод преобразует изображение облака слов в массив `NumPy` для дальнейшего использования с помощью `matplotlib`.

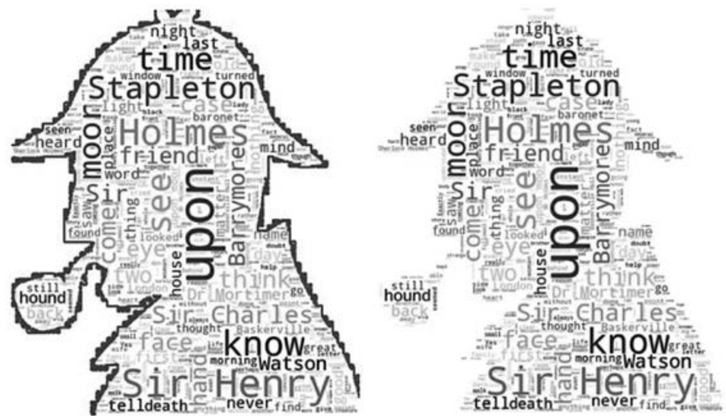


Рис. 3.4. Пример облака слов в маске с контуром (*слева*) и без (*справа*)

Отрисовка облака слов

Код листинга 3.10 добавляет облаку слов заголовок и использует `matplotlib` для его отображения. Он также сохраняет изображение облака слов в виде файла.

Листинг 3.10. Отображение и сохранение облака слов

wc_hound.py, part 3

```
plt.figure()
plt.title("Chamberlain Hunt Academy Senior Class Presents:\n",
          fontsize=15, color='brown')
plt.text(-10, 0, "The Hound of the Baskervilles",
        fontsize=20, fontweight='bold', color='brown')
plt.suptitle("7:00 pm May 10-12 McComb Auditorium",
             x=0.52, y=0.095, fontsize=15, color='brown')
plt.imshow(colors, interpolation="bilinear")
plt.axis('off')
plt.show()
##plt.savefig('hound_wordcloud.png')
```

В начале инициализируем рисунок `matplotlib`, затем вызываем метод `title()`, в который передаем название школы («Chamberian Hunt Academy Senior Class presents»), а также размер шрифта и его цвет.

Нам нужно, чтобы название спектакля было крупнее и жирнее других заголовков. Поскольку с помощью `matplotlib` изменить стиль текста в строке нельзя, мы используем метод `text()` для определения нового заголовка. В него передаем

координаты (x, y) согласно осям рисунка, текстовую строку и детали стиля текста. Оптимальные координаты для размещения текста мы подбираем методом проб и ошибок. При использовании изображения Холмса с iStock вам может потребоваться изменить координату x с `-10` на другое значение, чтобы добиться гармонии с асимметричным силуэтом.

Завершаем заголовки, указывая время и место проведения спектакля в нижней части рисунка. Можно снова использовать метод `text()`, но вместо этого мы воспользуемся его альтернативой — методом `suptitle()` из коллекции функций `pyplot`. Название метода означает «суперзаголовки». Передаем ему текст, координаты рисунка (x, y) и детали стилизации.

Для отображения облака слов вызываем `imshow()`, чтобы показать изображение, и передаем ему ранее созданный массив `colors`. Для интерполяции цветов указываем `bilinear`.

Отключаем оси рисунка и отображаем облако слов через вызов `show()`. Если вы хотите сохранить рисунок, раскомментируйте метод `savefig()`. Имейте в виду, что `matplotlib` может считывать расширение в имени файла и сохранять рисунок в верном формате. Как уже говорилось, команда сохранения не будет выполняться, пока вы вручную не закроете рисунок.

Настройка облака слов

Код листинга 3.10 создает облако слов, показанное на рис. 3.5. У вас может получиться иное расположение слов, поскольку алгоритм здесь стохастический.

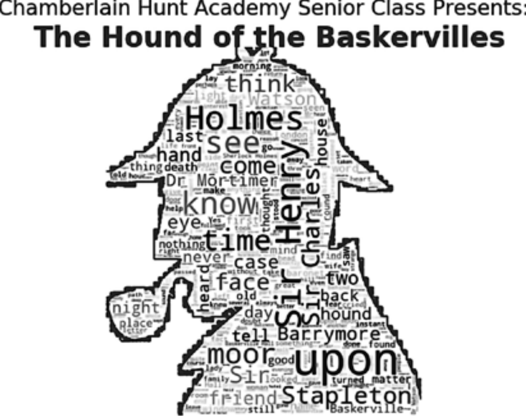


Рис. 3.5. Флаер, сгенерированный кодом wc_hound.py

Размер изображения можно изменить, добавив аргумент при инициализации рисунка. Вот пример: `plt.figure(figsize=(50, 60))`.

Известно и множество других способов изменить результат. Например, если установить параметр `margin` равным `10`, то облако слов получится менее плотным (рис. 3.6).

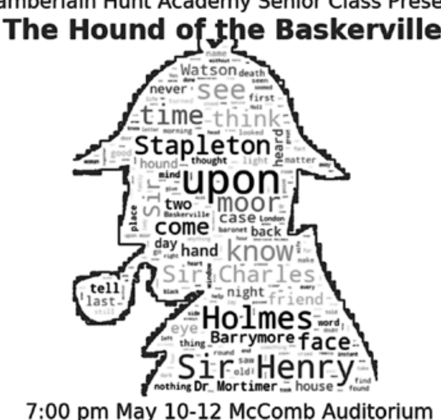


Рис. 3.6. Облако слов, сгенерированное с margin=10

Изменение параметра `random_state` также изменит расположение слов в маске (рис. 3.7).

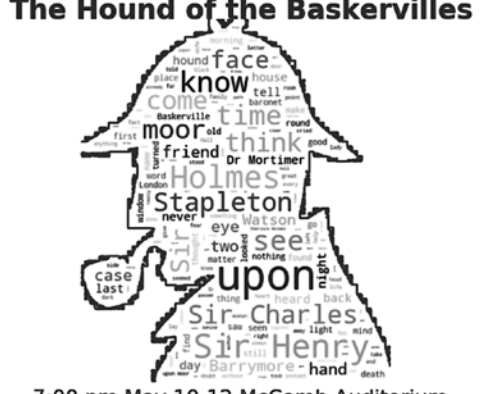


Рис. 3.7. Облако слов, сгенерированное с margin=10 и random_state=6

Настройка параметров `max_words` и `relative_scaling` также влияет на представление облака слов. В зависимости от нужного вам уровня детализации все эти вариации могут как помочь, так и навредить.

Итоги

В этой главе мы использовали техники экстрактивной суммаризации, чтобы создать краткий обзор речи Мартина Лютера Кинга-младшего «У меня есть мечта». Затем мы использовали бесплатный готовый модуль `gensim` для суммаризации речи адмирала Макрейвена, что потребовало еще меньшего количества кода. В завершение же мы применили модуль `wordcloud` для создания интересного дизайна с использованием облака слов.

Дополнительная литература

Книга «Automate the Boring Stuff with Python: Practical Programming for Total Beginners»[1] (No Starch Press, 2015) Эла Свейгарта (Al Sweigart) раскрывает тему регулярных выражений (глава 7) и веб-скрапинга (глава 11), включая использование модулей `requests` и Beautiful Soup.

«Make Your Bed: Little Things That Can Change Your Life... And Maybe the World»[2], 2nd ed. (Grand Central Publishing, 2017) Уильяма Макрейвена (William H. McRaven) — книга для по саморазвития, в основу которой легло напутственное обращение адмирала в Техасском университете. Саму речь вы также можете найти онлайн на https://www.youtube.com/.

Усложняем проект: ночные игры

Используйте `wordcloud` для создания новой игры в рамках сюжета «Ночных игр». Сделайте суммаризацию описаний фильмов из Википедии или IMDb и узнайте, смогут ли ваши друзья угадать название кино. На рис. 3.8 показаны некоторые примеры.

Если вы не особо увлекаетесь кино, то выберите другую сферу. В качестве альтернативы можно взять известные романы, эпизоды «Звездного пути» («Star Trek») или тексты песен (рис. 3.9).

[1] Свейгарт Э. «Автоматизация рутинных задач с помощью Python: практическое руководство для начинающих».

[2] Макрейвен У. «Заправляй кровать. 10 простых правил, которые помогут изменить твою жизнь и, возможно, весь мир».

Рис. 3.8. Облака слов для двух фильмов, вышедших в 2010 году: «Как приручить дракона» («How to Train Your Dragon») и «Принц Персии» («Prince of Persia»)

Рис. 3.9. Облако слов, созданное из текста песни «I.G.Y.» Дональда Фейгена

В последние годы вновь становятся популярными настольные игры, так что вы можете поддержать этот тренд и напечатать облака слов на карточках. Возможен

и альтернативный вариант — в рамках цифрового пространства: дайте игроку несколько вариантов ответов для каждого облака слов. При этом игра должна отслеживать количество верных ответов.

Усложняем проект: суммаризация суммаризаций

Протестируйте свою программу из проекта 3 на ранее суммаризованном тексте, например страницах Википедии. Всего пять предложений сформировали хороший обзор gensim.

```
Ввести максимальное количество слов в предложении для суммаризации: 30
Ввести количество предложений в резюме: 5

СУММАРИЗАЦИЯ:
Gensim is implemented in Python and Cython.
Gensim is an open-source library for unsupervised topic modeling and natural
language processing, using modern statistical machine learning.
[12] Gensim is commercially supported by the company rare-technologies.com,
who also provide student mentorships and academic thesis projects for Gensim
via their Student Incubator programme.
The software has been covered in several new articles, podcasts and
interviews.
Gensim is designed to handle large text collections using data streaming and
incremental online algorithms, which differentiates it from most other machine
learning software packages that target only in-memory processing.
```

Далее попробуйте применить версию gensim из проекта 4 к скучным соглашениям на обслуживание, которые никто никогда не читает. Пример соглашения об использовании служб Microsoft доступен по адресу https://www.microsoft.com/en-us/servicesagreement/default.aspx[1]. Конечно же, чтобы оценить результаты, вам придется прочитать все соглашение, чего практически никто не делает. Такой вот замкнутый круг!

Усложняем проект: суммаризация повести

Напишите программу, которая делает суммаризацию романа «Собака Баскервилей» по главам. Пусть резюме глав будут краткими — около 75 слов каждое.

Для получения копии повести с названиями глав выполните скрапинг текста с сайта Project Gutenberg при помощи следующей строки кода:

```
url = 'http://www.gutenberg.org/files/2852/2852-h/2852-h.htm'.
```

[1] https://www.microsoft.com/ru-ru/servicesagreement/ — на русском языке.

Чтобы разделить элементы глав, а не абзацев, используйте такой код:

```
chapter_elems = soup.select('div[class="chapter"]')
chapters = chapter_elems[2:]
```

Вам также понадобится выбрать элементы абзацев (`p_elems`) из каждой главы, используя тот же способ, что и в dream_summary.py.

Сниппеты ниже показывают некоторые результаты при использовании 75 слов для суммаризации каждой главы:

```
--snip--

Chapter 3:
"Besides, besides—" "Why do you hesitate?" "There is a realm in which the
most
acute and most experienced of detectives is helpless." "You mean that the
thing is supernatural?" "I did not positively say so." "No, but you evidently
think it." "Since the tragedy, Mr. Holmes, there have come to my ears several
incidents which are hard to reconcile with the settled order of Nature." "For
example?" "I find that before the terrible event occurred several people had
seen a creature upon the moor which corresponds with this Baskerville demon,
and which could not possibly be any animal known to science.

--snip--

Chapter 6:
"Bear in mind, Sir Henry, one of the phrases in that queer old legend which
Dr. Mortimer has read to us, and avoid the moor in those hours of darkness
when the powers of evil are exalted." I looked back at the platform when we
had left it far behind and saw the tall, austere figure of Holmes standing
motionless and gazing after us.

Chapter 7:
I feared that some disaster might occur, for I was very fond of the old man,
and I knew that his heart was weak." "How did you know that?" "My friend
Mortimer told me." "You think, then, that some dog pursued Sir Charles, and
that he died of fright in consequence?" "Have you any better explanation?" "I
have not come to any conclusion." "Has Mr. Sherlock Holmes?" The words took
away my breath for an instant but a glance at the placid face and steadfast
eyes of my companion showed that no surprise was intended.

--snip--
Chapter 14:
"What's the game now?" "A waiting game." "My word, it does not seem a very
cheerful place," said the detective with a shiver, glancing round him at the
gloomy slopes of the hill and at the huge lake of fog which lay over the
Grimpen Mire.

Far away on the path we saw Sir Henry looking back, his face white in the
moonlight, his hands raised in horror, glaring helplessly at the frightful
thing which was hunting him down.

--snip--
```

Усложняем проект: важно не только что ты говоришь, но и как!

Написанные вами на данный момент программы суммаризации текста выводят предложения строго по *степени их важности*. Это означает, что последнее предложение речи (или любого текста) в резюме может стать первым. Цель суммаризации — находить важные предложения, но ничто не мешает нам изменить порядок их отображения.

Напишите программу суммаризации текста, отображающую наиболее важные предложения в исходном *порядке их появления*. Сравните результаты с результатами программы в проекте 3. Удалось ли таким образом заметно улучшить резюме?

4

Отправка суперсекретных сообщений с помощью книжного шифра

«Ключ к Ребекке» — это роман-бестселлер Кена Фоллетта, который получил широкое признание критиков. Его действие происходит в Каире во время Второй мировой войны и основано на реальных событиях. Сюжет повествует о преследовании нацистского шпиона офицером британской разведки. Название романа относится к использованной шпионом системе шифрования, в которой ключом выступал известный готический роман «Ребекка», написанный Дафной дю Морье. «Ребекка» считается одним из величайших романов XX века, и германские военные действительно использовали его во время войны в качестве кодовой книги.

Шифр «Ребекка» представляет собой разновидность *одноразового шифровального блокнота*, невзламываемой техники шифрования, когда размер требуемого ключа не меньше самого отправляемого сообщения. Отправитель и получатель располагают копией блокнота, верхнюю страницу которого после однократного использования вырывают и уничтожают.

Одноразовые блокноты обеспечивают абсолютную безопасность — взломать такой шифр не получится даже с помощью квантового компьютера. Несмотря

на это, у таких блокнотов есть ряд недостатков, которые не позволяют использовать их повсеместно. Основные проблемы — это необходимость безопасно транспортировать и доставлять блокноты отправителю и получателю, хранить их, а также ручное кодирование и декодирование сообщений.

В «Ключе к Ребекке» для использования шифра обе стороны должны знать правила шифрования и иметь в наличии одинаковую редакцию книги. В этой главе мы преобразуем описанный в романе ручной метод в более безопасную и удобную в использовании цифровую технику. Вы поработаете с полезными функциями из Python Standard Library и модулями `collections` и `random`. Вы также немного больше узнаете о Юникоде — стандарте, используемом для обеспечения универсальной совместимости таких символов, как буквы и числа, между всеми платформами, устройствами и приложениями.

Одноразовый блокнот

Одноразовый блокнот (OTP, One-Time-Pad) — это упорядоченный набор листов с напечатанными на них случайными числами, обычно объединенными в группы по пять (рис. 4.1). Чтобы блокноты было удобно прятать, их делают такими маленькими, что иногда без увеличительного стекла и не прочтешь. Несмотря на архаичный вид, такие блокноты позволяют создавать самые безопасные шифры в мире, поскольку каждая буква шифруется уникальным ключом. В результате техники криптоанализа, например анализ частоты появления букв, для них просто не работают.

```
73983  91543  74556  01283
24325  88622  92061  02865
22764  47630  14408  80067
13154  81950  11992  84763
46381  99463  49155  40241
98484  77841  03878  14645
11774  73919  83946  40337
12396  26327  76612  12471
18432  41657  93893  10041
77381  39150  47951  83242
  211  02998  15002  08183
```

Рис. 4.1. Пример страницы одноразового блокнота

Чтобы зашифровать сообщение с помощью одноразового блокнота на рис. 4.1, мы начнем с присвоения каждой букве алфавита числа из двух цифр. *A* будет равна 01, *B* — 02 и т. д., как показано в таблице ниже.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Далее преобразуем буквы короткого сообщения в числа:

H	E	R	E		K	I	T	T	Y		K	I	T	T	Y	*Оригинальное сообщение*
08	05	18	05		11	09	20	20	25		11	09	20	20	25	*Буквы, преобразованные в числа*

Начиная с верхней левой части страницы и читая слева направо, мы присваиваем числовую пару (*ключ*) каждой букве и прибавляем ее к числовому значению этой буквы. Вы будете работать с парами чисел с основанием 10, поэтому при получении суммы больше 100 используйте модульную арифметику для усечения значения до последних двух цифр (103 станет 03). Числа в затененных ячейках таблицы ниже представляют результат модульной арифметики.

H	E	R	E		K	I	T	T	Y		K	I	T	T	Y	*Оригинальное сообщение*
08	05	18	05		11	09	20	20	25		11	09	20	20	25	*Буквы, преобразованные в числа*
73	98	39	15		43	74	55	60	12		83	24	32	58	86	*Из OTP отправителя*
81	03	57	20		54	83	75	80	37		94	33	52	78	11	*Криптограмма*

Последняя строка в таблице представляет криптограмму. Обратите внимание, что символы KITTY, повторяющиеся в чистом тексте, не повторяются в зашифрованном. Каждый шифр KITTY уникален.

Чтобы расшифровать криптограмму обратно в обычный текст, получатель использует такую же страницу из идентичного одноразового блокнота. Он располагает свои числовые пары под парами криптограммы и производит вычитание. Если получается отрицательное число, используется вычитание по модулю (прибавление 100 к значению криптограммы перед вычитанием). В завершение полученные числовые пары преобразуются обратно в буквы.

81	03	57	20		54	83	75	80	37		94	33	52	78	11	*Криптограмма*
73	98	39	15		43	74	55	60	12		83	24	32	58	86	*Из OTP получателя*
08	05	18	05		11	09	20	20	25		11	09	20	20	25	*Преобразование числа в буквы*
H	E	R	E		K	I	T	T	Y		K	I	T	T	Y	*Расшифрованный текст*

Чтобы гарантировать отсутствие повторов ключей, количество букв в сообщении не может превышать количество ключей в блокноте. Это вынуждает использовать короткие сообщения, которые проще шифровать/дешифровать и которые дают криптоаналитику меньше возможностей расшифровать сообщение. Помимо этих, есть и другие инструкции:

- Прописывать числа словами (к примеру, TWO для 2).
- Заканчивать предложения символом *X* вместо точки (к примеру, CALL AT NOONX).
- Прописывать словами любую другую пунктуацию, которой нельзя избежать (к примеру, COMMA).
- В конце исходного сообщения ставить *XX*.

Шифр «Ребекка»

В книге «Ключ к Ребекке» нацистский шпион использует разновидность одноразового блокнота. Идентичные редакции романа «Ребекка» приобретаются в Португалии. Два экземпляра шпион оставляет у себя, и еще два отправлены штабу фельдмаршала Роммеля в Северной Африке. Зашифрованные сообщения передаются по радио на заранее заданной частоте. В день отправляется не более одного сообщения, причем сеанс связи происходит строго в полночь.

Чтобы использовать ключ, шпион берет текущую дату — скажем, 28 мая 1942 года, и складывает день с годом (28 + 42 = 70). Так участники определяют, какую страницу романа нужно использовать в качестве страницы блокнота. Поскольку май — это пятый месяц в году, каждое пятое слово в предложении не учитывается. Так как использовать шифр «Ребекка» планировалось только относительно короткий промежуток времени в 1942 году, шпиону не нужно было беспокоиться о повторах в календаре, которые могли вызвать повторы в ключах.

Первым сообщением шпиона было: HAVE ARRIVED. CHECKING IN. ACKNOWLEDGE («Прибыл. Регистрируюсь. Подтвердите получение»). Начиная с верхней части страницы 70, он продолжал читать, пока не встретил букву *H*. Это был 10-й символ без учета каждой пятой буквы. Десятой в алфавите идет буква *J*, поэтому в криптограмме для представления буквы *H* он использовал именно ее. Следующая буква, *A*, шла через три буквы после *H*, значит, для ее шифра использовалась третья буква алфавита, *C*. Так продолжалось, пока не было зашифровано все сообщение. Как утверждает автор романа, к редким буквам, таким как *X* и *Z*, применялись особые правила, но сами правила он не описывает.

Использование книги имело значительное преимущество перед реальным одноразовым блокнотом. Как говорил Фоллетт: «По блокноту сразу можно понять, что он используется для шифрования, а вот книга выглядит безобидно». Однако не обошлось и без недостатков: процесс шифрования/дешифрования очень утомителен и легко было допустить ошибку. Посмотрим, удастся ли нам исправить это с помощью Python.

Проект #6. Цифровой ключ к «Ребекке»

Превращение техники «Ребекки» в цифровую программу дает несколько преимуществ по сравнению с одноразовым блокнотом.

- Процессы кодирования/декодирования станут быстрыми и безошибочными.
- Появится возможность отправлять более длинные сообщения.
- Можно непосредственно шифровать точки, запятые и даже пробелы.
- Редкие буквы, например *z*, можно выбирать из любой части книги.
- Кодовую книгу легко спрятать среди тысяч электронных на диске или в облаке.

Последний пункт особенно важен. В романе офицер британской разведки находит копию книги «Ребекка» на захваченной заставе немцев. Путем простой дедукции он понимает, что это альтернатива одноразовому блокноту. В случае цифрового варианта это было бы намного сложнее. Роман можно было бы хранить на небольшом устройстве вроде SD-карты, которую легко спрятать.

Это сближает книгу с одноразовым блокнотом, который зачастую не превышает размера почтовой марки.

И все же у цифрового подхода есть один недостаток: программу можно *обнаружить*. Если при использовании одноразового блокнота шпиону приходилось просто запомнить правила, то в цифровом случае правила должны быть скрыты в ПО. Это слабое место можно свести к минимуму, написав программу так, чтобы она выглядела безобидно — или хотя бы неоднозначно — и запрашивала от пользователя ввод сообщения и названия кодовой книги.

ЗАДАЧА

Написать программу на Python, шифрующую и дешифрующую сообщения, используя в качестве одноразового блокнота цифровой формат романа.

Стратегия

В отличие от шпиона вам не потребуются все описанные в романе правила, тем более что многие и так бы не сработали. Если вам доводилось пользоваться электронными книгами, то вы знаете, что нумерация страниц в них бессмысленна. Изменение размеров страниц и шрифта лишает номера страниц уникальности. А так как вы можете выбирать буквы из любой части книги, то уже не потребуются никакие особые правила для редких букв или пропуска чисел при подсчете.

Итак, вам не нужно стремиться к идеальному воссозданию шифра «Ребекка». Достаточно создать нечто подобное, но желательно лучше.

К нашему удобству, *итерируемые объекты* Python, такие как списки и кортежи, отслеживают каждый свой элемент с помощью числовых индексов. Скачав роман в виде списка, можно использовать эти индексы в качестве уникальных стартовых ключей для каждой главы. Далее можно сместить индексы на основе дня года, сымитировав метод из романа «Ключ к Ребекке».

К сожалению, «Ребекка» пока еще не находится в публичном доступе. Вместо нее мы задействуем роман Артура Конан Дойла «Затерянный мир», который использовали в главе 2. В этом произведении содержится 51 уникальный символ, который встречается 421 545 раз, а значит, индексы можно выбирать случайно с очень небольшим шансом повтора. То есть при каждой шифровке сообщения вы сможете использовать в роли одноразового блокнота всю книгу, а не ограничиваться небольшой коллекцией чисел с одной странички.

ПРИМЕЧАНИЕ

При желании можете скачать и использовать цифровую версию «Ребекки», просто я не могу предоставить вам ее бесплатно.

Поскольку вы будете повторно использовать одну и ту же книгу, то нужно подумать о повторении ключей *между разными сообщениями*, а также *внутри сообщения*. Чем длиннее сообщение, тем больше материала может обработать криптоаналитик, а значит, и взломать код проще. И если каждое сообщение отправлять с одним и тем же ключом шифрования, то все перехваченные сообщения можно рассматривать как одно большое целое.

Для передачи каждого сообщения можно сымитировать действия шпиона и смещать номера индексов по дням года, используя диапазон от 1 до 366, чтобы учесть високосные годы. В подобной схеме 1 февраля будет представлять число 32. Таким образом, книга каждый раз будет превращаться в новый одноразовый блокнот, поскольку для одних и тех же символов будут использоваться разные ключи. Инкремент на один или более сбрасывает все индексы и как бы

вырывает предыдущую страницу. Здесь, в отличие от одноразового блокнота, вам не нужно беспокоиться об утилизации вырванной бумажки.

Что же касается проблемы повторения внутри сообщения, то можно перед его отправкой выполнять проверку. В программе, хотя это и маловероятно, один и тот же ключ в процессе шифрования может быть выбран дважды, тем самым один и тот же индекс будет использован дважды. Дублирующиеся индексы — это, по сути, повторяющиеся ключи, которые помогут криптоаналитику взломать ваш код. Поэтому при обнаружении повторяющихся индексов можно либо перезапустить программу, либо перефразировать сообщение.

Вам также понадобятся правила, аналогичные использованным в «Ключе к Ребекке».

- Обе стороны должны располагать одинаковыми копиями «Затерянного мира».
- Обе стороны должны знать, как смещаются индексы.
- Сообщения необходимо формулировать максимально сжато.
- Числа следует прописывать словами.

Код для шифрования

Приведенный ниже код rebecca.py будет получать сообщение и возвращать его зашифрованную или текстовую версию согласно установке пользователя. Сообщение можно набрать или скачать с сайта книги. Вам также потребуется текстовый файл lost.txt, расположенный в одном каталоге с кодом.

Для ясности мы будем использовать имена переменных, такие как `ciphertext`, `encrypt`, `message` и т. д. Хотя будь мы реальными шпионами, то избегали бы столь явных названий на случай, если противник получит доступ к нашему ноутбуку.

Импорт модулей и определение функции main()

Код листинга 4.1 импортирует модули и определяет функцию `main()`, используемую для запуска программы. Эта функция запрашивает пользовательский ввод, вызывает функции, необходимые для шифрования/дешифрования текста, проверяет наличие повторяющихся ключей и выводит либо криптограмму, либо расшифрованный текст.

Определить `main()` можно как в начале, так и в конце программы. Иногда она предоставляет хорошее и легкочитаемое содержимое всей программы. В других случаях она может выглядеть не к месту, как повозка впереди лошади. С точки зрения Python неважно, где именно вы ее разместите, при условии вызова этой функции в конце.

Листинг 4.1. Импорт модулей и определение функции main()

rebecca.py, часть 1

```
import sys
import os
import random
from collections import defaultdict, Counter
def main():
    message = input("Enter plaintext or ciphertext: ")
    process = input("Enter 'encrypt' or 'decrypt': ")
    while process not in ('encrypt', 'decrypt'):
        process = input("Invalid process. Enter 'encrypt' or 'decrypt': ")
    shift = int(input("Shift value (1-366) = "))
    while not 1 <= shift <= 366:
        shift = int(input("Invalid value. Enter digit from 1 to 366: ")
  ❶ infile = input("Enter filename with extension: ")

    if not os.path.exists(infile):
        print("File {} not found. Terminating.".format(infile),
              file=sys.stderr)
        sys.exit(1)
    text = load_file(infile)
    char_dict = make_dict(text, shift)

    if process == 'encrypt':
        ciphertext = encrypt(message, char_dict)
      ❷ if check_for_fail(ciphertext):
            print("\nProblem finding unique keys.", file=sys.stderr)
            print("Try again, change message, or change code book.\n",
                  file=sys.stderr)
            sys.exit()
      ❸ print("\nCharacter and number of occurrences in char_dict: \n")
        print("{: >10}{: >10}{: >10}".format('Character', 'Unicode',
                                             'Count'))
        for key in sorted(char_dict.keys()):
            print('{:>10}{:>10}{:>10}'.format(repr(key)[1:-1],
                                              str(ord(key)),
                                              len(char_dict[key])))

        print('\nNumber of distinct characters: {}'.format(len(char_dict)))
        print("Total number of characters: {:,}\n".format(len(text)))

        print("encrypted ciphertext = \n {}\n".format(ciphertext))
        print("decrypted plaintext = ")

      ❹ for i in ciphertext:
            print(text[i - shift], end='', flush=True)

    elif process == 'decrypt':
        plaintext = decrypt(message, text, shift)
        print("\ndecrypted plaintext = \n {}".format(plaintext))
```

Начинаем с импорта `sys` и `os`, двух модулей, которые позволят взаимодействовать с операционной системой. Далее импортируем модуль `random`, а затем `defaultdic` и `Counter` из модуля `collections`.

Модуль `collections` является частью Python Standard Library и включает несколько типов данных контейнера. С помощью `defaultdic` можно собирать словарь на лету. Если `defaultdic` встречает недостающий ключ, он не выдает ошибку, а подставляет значение по умолчанию. С его помощью мы будем создавать словарь из символов «Затерянного мира» и соответствующих им значений индексов.

`Counter` — это подкласс словаря для подсчета хешируемых объектов. Здесь элементы хранятся в качестве ключей словаря, а их количество — в качестве значений. С его помощью мы будем проверять криптограмму, чтобы убедиться в отсутствии повторения индексов.

Далее следует определение функции `main()`. Она начинается с запроса от пользователя сообщения для шифрования/дешифрования. Для максимальной безопасности пользователь должен это сообщение вводить вручную. Далее программа просит его указать требуемое действие — шифрование или дешифрование. После выбора программа запросит значение `shift`. Это значение представляет день года в диапазоне от 1 до 366 включительно. Затем идет запрос `infile`, которым будет lost.txt, цифровая версия «Затерянного мира» ❶.

Прежде чем продолжать, программа проверяет наличие этого файла. Для этого она использует метод `path_exists()` модуля операционной системы, передавая ему переменную `infile`. Если файла не существует или путь и/или имя файла указаны неверно, программа сообщает об этом пользователю, используя опцию `file=sys.stderr` для окрашивания сообщения красным в оболочке Python, и завершается командой `sys.exit(1)`. Значение `1` указывает, что программа завершилась с ошибкой, а не корректным образом.

Далее идет вызов нескольких функций, которые мы определим позднее. Первая из них загружает файл lost.txt в виде строки `text`, которая включает небуквенные символы, такие как пробелы и знаки препинания. Вторая функция создает словарь из символов и соответствующих им индексов с применением значения смещения.

Теперь прописываем условие для оценки используемого процесса. Как я уже сказал, для ясности мы используем явные названия — `encrypt` и `decrypt`. В реальной шпионской деятельности этого лучше избегать. Если пользователь выбрал шифрование, вызываем функцию, шифрующую сообщение с помощью словаря символов. По возвращении функцией результата можно считать шифрование выполненным. Но не стоит уповать, что все пройдет без запинки. Нужно

убедиться в верности дешифровки и отсутствии повторяющихся ключей. Для этого мы начинаем серию действий для контроля качества.

Сначала проверяем, нет ли повтора ключей ❷. Если эта функция вернет `True`, просим пользователя повторить попытку, изменив сообщение или избранную для шифрования книгу. Для каждого символа в сообщении мы используем `char_dict` и выбираем индекс случайно. Даже при наличии для каждого символа сотни или тысячи индексов, все равно есть шанс выбрать один и тот же индекс более одного раза.

Повторное выполнение программы с немного измененными параметрами, как я рассказывал ранее, должно исправить проблему, если только шифруется недлинное сообщение с большим количеством низкочастотных символов. Для обработки такого редкого случая может потребоваться перефразировать сообщение или найти более обширный текст, нежели «Затерянный мир».

ПРИМЕЧАНИЕ

В Python модуль `random` производит лишь псевдослучайные числа, которые можно спрогнозировать. Любой шифр, использующий псевдослучайные числа, потенциально может быть взломан. Для максимальной безопасности при генерации случайных чисел следует использовать функцию Python `os.urandom()`.

Теперь выведем содержимое словаря символов, чтобы вы могли увидеть, сколько раз различные символы встречаются в романе ❸. Это поможет строить сообщения, хотя в «Затерянном мире» пригодных символов очень много.

```
Символ и количество вхождений в char_dict:

    Символ    Юникод    Количество
        \n        10          7865
                  32         72185
         !        33           282
         "        34          2205
         '        39           761
         (        40            62
         )        41            62
         ,        44          5158
         -        45          1409
         .        46          3910
         0        48             1
         1        49             7
         2        50             3
         3        51             2
         4        52             2
         5        53             2
         6        54             1
```

```
         7       55          4
         8       56          5
         9       57          2
         :       58         41
         ;       59        103
         ?       63        357
         a       97      26711
         b       98       4887
         c       99       8898
         d      100      14083
         e      101      41156
         f      102       7705
         g      103       6535
         h      104      20221
         i      105      21929
         j      106        431
         k      107       2480
         l      108      13718
         m      109       8438
         n      110      21737
         o      111      25050
         p      112       5827
         q      113        204
         r      114      19407
         s      115      19911
         t      116      28729
         u      117      10436
         v      118       3265
         w      119       8536
         x      120        573
         y      121       5951
         z      122        296
         {      123          1
         }      125          1

Количество уникальных символов: 51
Общее количество символов: 421 545
```

При генерации этой таблицы мы используем Format Specification Mini-Language (https://docs.python.org/3/library/string.html#formatspec), с помощью которого выводим заголовки для трех столбцов. Число в фигурных скобках указывает, сколько символов должно содержаться в строке, а знак «больше» означает выравнивание по правому краю.

Затем программа перебирает ключи в словаре символов и выводит их, используя ту же ширину столбцов и выравнивание. Она выводит символ, значение Юникода и количество вхождений этого символа в тексте.

Для вывода ключа используется `repr()`. Эта встроенная функция возвращает строку, содержащую выводимое представление объекта. То есть она

возвращает всю информацию об объекте в формате, пригодном для отладки и разработки. Это позволяет явно выводить символы вроде перевода строки `\n` и пробела. Диапазон индексов `[1:-1]` исключает из строки вывода кавычки с обеих сторон.

Встроенная функция `ord()` возвращает целое число, представляющее кодовую точку Юникода для символа. Компьютеры работают только с числами, поэтому необходимо присваивать число всем возможным символам, таким как *%*, *5*, ☺ или *А*. *Стандарт Юникод* гарантирует использование уникального числа и универсальную совместимость для каждого символа независимо от платформы, устройства, приложения или языка. Показывая пользователю значения Юникода, программа позволяет ему уловить любые странности в рамках текстового файла, например отображение одной и той же буквы в виде нескольких различных символов.

Для третьего столбца мы получаем длину каждого ключа словаря. Она представляет количество вхождений каждого символа в романе. Затем программа выводит количество различных символов и общее их число в тексте.

Процесс шифрования завершается выводом криптограммы, а затем дешифрованного открытого текста для проверки. Для дешифрования сообщения программа перебирает каждый элемент криптограммы и использует его в качестве индекса для `text` ❹, вычитая ранее добавленное значение `shift`. При выводе результатов программа использует `end=''` вместо предустановленного перевода строки, поэтому каждый символ не оказывается на отдельной строке.

Завершается функция `main()` условной инструкцией для проверки `process == 'decrypt'`. Если пользователь выбирает дешифрование сообщения, то программа вызывает функцию `decrypt()`, после чего выводит дешифрованный текст. Заметьте, что здесь можно просто использовать `else`, но я выбрал `elif`, что обеспечивает лучшую ясность и читаемость.

Загрузка файла и создание словаря

В листинге 4.2 мы определим функции для загрузки текстового файла и создания словаря из содержащихся в файле символов и соответствующих им индексов.

Листинг 4.2. Определение функций load_file() и make_dict()

rebecca.py, часть 2

```
def load_file(infile):
    """Прочитаем и вернем текстовый файл в виде строки из символов нижнего
    регистра."""
    with open(infile) as f:
        loaded_string = f.read().lower()
    return loaded_string
```

```
❶ def make_dict(text, shift):
      """Вернем словарь символов и сдвинутых индексов."""
      char_dict = defaultdict(list)
      for index, char in enumerate(text):
        ❷ char_dict[char].append(index + shift)
      return char_dict
```

Начинается листинг с определения функции для загрузки текста в виде строки. Использование `with` для его открытия гарантирует, что по завершении функции он автоматически закроется.

Некоторые пользователи при загрузке текстовых файлов могут получить, например, такую ошибку:

```
UnicodeDecodeError: 'charmap' codec can't decode byte 0x81 in position
27070:character maps to <undefined>
```

В этом случае попробуйте изменить функцию `open`, добавив аргументы `encoding` и `errors`.

```
with open(infile, encoding='utf-8', errors='ignore') as f:
```

Подробнее об этой проблеме я рассказываю на с. 66 главы 2.

После открытия файла считываем его как строку и преобразуем весь текст в нижний регистр. Затем возвращаем эту строку.

Следующим шагом преобразуем полученную строку в словарь. Определяем функцию, которая в качестве аргументов получает эту строку и значение `shift` ❶. Программа с помощью `defaultdict()` создает переменную `char_dict`, которая выступает в качестве словаря. Далее в `defaultdict()` передаем конструктор типа для `list`, так как нам нужно, чтобы значения словаря были представлены списком индексов.

При использовании `defaultdict()` каждый раз, когда операция встречает элемент, еще отсутствующий в словаре, вызывается функция `default_factory()` без аргументов, а ее вывод используется в качестве значения. Любой несуществующий ключ получает значение, возвращаемое `default_factory()`, и `KeyError` не возникает.

Если же попробовать создать словарь на лету, без вспомогательного модуля `collections`, то возникнет `KeyError`, как демонстрирует пример ниже.

```
>>> mylist = ['a', 'b', 'c']
>>> d = dict()
>>> for index, char in enumerate(mylist):
    d[char].append(index)
```

```
Traceback (most recent call last):
 File "<pyshell#16>", line 2, in <module>
  d[char].append(index)
KeyError: 'a'
```

Встроенная функция `enumerate()` действует как автоматический счетчик, так что мы легко получаем индекс для каждого символа в строке, сгенерированной из «Затерянного мира». Ключи в `char_dict` являются символами, а символы могут встречаться в `text` тысячи раз. Получается, что значения словаря — списки, которые хранят индексы для всех этих вхождений символов. Прибавляя значение смещения к индексу при его внесении в список значений, мы гарантируем уникальность индексов для каждого сообщения ❷.

Завершается функция возвращением словаря символов.

Шифрование сообщения

В листинге 4.3 мы определяем функцию для шифрования сообщения и получаем криптограмму, которая представляет список индексов.

Листинг 4.3. Определение функции для шифрования сообщения

rebecca.py, часть 3

```
def encrypt(message, char_dict):
    """Вернуть список индексов, представляющих символы в сообщении"""
    encrypted = []
    for char in message.lower():
      ❶ if len(char_dict[char]) > 1:
            index = random.choice(char_dict[char])
        elif len(char_dict[char]) == 1: # Random.choice не работает,
                                          если есть только 1 вариант
            index = char_dict[char][0]
      ❷ elif len(char_dict[char]) == 0:
            print("\nCharacter {} not in dictionary.".format(char),
                  file=sys.stderr)
            continue
    encrypted.append(index)
return encrypted
```

Функция `encrypt()` получает в качестве аргументов сообщение и `char_dict`. Начинается она с создания пустого списка для хранения криптограммы. Далее выполняются перебор символов в `message` и их перевод в нижний регистр для соответствия символам в `char_dict`.

Если количество связанных с символами индексов окажется больше 1, программа использует метод `random.choice()` для случайного выбора одного из индексов символа ❶.

Если символ встречается в `char_dict()` всего один раз, то `random.choice()` выдаст ошибку. Поэтому программа использует условие и жестко фиксирует выбор индекса, который находится в позиции `[0]`.

Если какой-то символ в «Затерянном мире» не используется, в словарь он не попадет, поэтому предусмотрена соответствующая проверка — условием ❷. Если результат оценивается как `True`, пользователь получает предупреждение, и посредством `continue` программа возвращается к началу цикла без выбора индекса. Позже, когда контроль качества криптограммы будет выполнен, в дешифрованном тексте на месте пропущенного символа появится пробел.

Если `continue` не вызывается, программа добавляет индекс в список `encrypted`. Когда цикл завершается, мы возвращаем список, завершая функцию.

Чтобы увидеть, как это работает, рассмотрим первое сообщение, которое нацистский шпион отправляет в романе «Ключ к Ребекке»:

HAVE ARRIVED. CHECKING IN. ACKNOWLEDGE.

Использование этого сообщения и значения смещения — 70 — привело к генерации следующей случайной криптограммы:

```
[125711, 106950, 85184, 43194, 45021, 129218, 146951, 157084, 75611, 122047,
121257, 83946, 27657, 142387, 80255, 160165, 8634, 26620, 105915, 135897,
22902, 149113, 110365, 58787, 133792, 150938, 123319, 38236, 23859, 131058,
36637, 108445, 39877, 132085, 86608, 65750, 10733, 16934, 78282]
```

Ввиду стохастической природы алгоритма ваш результат может отличаться.

Дешифрование сообщения

В листинге 4.4 определяется функция для дешифрования криптограммы. В ответ на запрос от функции `main()` пользователь будет копировать и вставлять криптограммы.

Листинг 4.4. Определение функции для дешифрования сообщения

rebecca.py, часть 4

```
def decrypt(message, text, shift):
    """Дешифруем список и возвращаем открытый текст"""
    plaintext = ''
    indexes = [s.replace(',', '').replace('[', '').replace(']', '')
               for s in message.split()]
    for i in indexes:
        plaintext += text[int(i) - shift]
    return plaintext
```

Листинг начинается с определения функции `decrypt()` с сообщением, романом (`text`) и значением `shift` в качестве параметров. Конечно же, сообщение

представлено в виде криптограммы, состоящей из списка чисел, выражающих смещенные индексы. Мы сразу же создаем пустую строку для хранения дешифрованного текста.

Большинство людей наверняка перекопируют криптограмму в ответ на запрос от функции `main()`. Этот ввод может содержать или не содержать квадратные скобки, которые есть в списке. А поскольку пользователь вводит криптограмму с помощью функции `input()`, результатом станет *строка*. Чтобы преобразовать индексы в целые числа, которые можно сместить, сначала следует удалить все нецифровые символы. Для этого применяем методы `stringreplace()` и `split()` при сопутствующем использовании спискового включения для возвращения списка. Списковое включение — это краткий способ выполнения циклов в Python.

Чтобы использовать `replace()`, мы передаем ему символ, который нужно удалить, и символ на замену. В нашем случае это пробел. Обратите внимание, что можно «объединить» их с помощью точечной нотации, обработав запятые и скобки в один заход. Круто?

Далее переходим к перебору индексов. Программа преобразует текущий индекс из строки в целое число, давая возможность вычесть значение смещения, примененное в процессе шифрования. Индекс мы используем, чтобы обратиться к списку символов и получить соответствующий символ. Далее мы добавляем этот символ в строку `plaintext` и возвращаем `plaintext`, когда цикл завершается.

Проверка на сбой и вызов функции main()

Код листинга 4.5 определяет функцию для проверки криптограммы на наличие повторяющихся индексов (ключей) и завершает программу вызовом функции `main()`. Если проверяющая функция обнаружит повторение индексов, значит, шифрование может оказаться уязвимым и функция `main()` сообщит пользователю, как это можно исправить, после чего завершится.

Листинг 4.5. Определение функции для проверки повторения индексов и вызова main()

rebecca.py, part 5

```
def check_for_fail(ciphertext):
    """Вернем True, если шифрованный текст содержит дубликаты ключей"""
    check = [k for k, v in Counter(ciphertext).items() if v > 1]
    if len(check) > 0:
        return True

if __name__ == '__main__':
main()
```

Здесь определяется функция `check_for_fail()`, получающая в качестве аргумента криптограмму, которую проверяет на наличие повторов индексов. Напомню, что вариант с одноразовым блокнотом работает благодаря тому, что каждый ключ уникален. Таким образом, каждый индекс в криптограмме должен быть уникален.

Для поиска повторов снова используем `Counter`. Программа задействует списковое включение для построения списка, содержащего все повторяющиеся индексы. Здесь `k` означает ключ (словаря), а `v` — значение (словаря). Поскольку `Counter` создает словарь количеств вхождений каждого ключа, то здесь мы указываем: для каждой пары «ключ — значение» в словаре, полученном из криптограммы, создать список всех ключей, встречающихся более одного раза. При наличии повторов добавить соответствующий ключ в список `check`.

Теперь следует лишь получить длину `check`. Если она окажется больше нуля, значит, шифрование уязвимо и программа возвращает `True`.

В завершение используем стандартный код для вызова программы в качестве модуля или в автономном режиме.

Отправка сообщений

Приведенное ниже сообщение основано на отрывке из книги «Ключ к Ребекке». Вы найдете его в доступном для скачивания каталоге Chapter_4 под именем allied_attack_plan.txt.

В качестве теста попробуйте отправить его со смещением 70. При получении от программы запроса на ввод текста используйте команды операционной системы «Выбрать все», «Скопировать» и «Вставить» для переноса текста. Если сообщение не пройдет проверку `check_for_fail()`, выполните ее еще раз.

```
Allies plan major attack for Five June. Begins at oh five twenty with
bombardment from Aslagh Ridge toward Rommel east flank. Followed by tenth
Indian Brigade infantry with tanks of twenty second Armored Brigade on Sidi
Muftah. At same time, thirty second Army Tank Brigade and infantry to charge
north flank at Sidra Ridge. Three hundred thirty tanks deployed to south and
seventy to north.
```

Эта технология хороша тем, что можно использовать правильную пунктуацию, по крайней мере при вводе сообщения в окне интерпретатора. Текст, скопированный извне, может потребовать очистки от символов перевода строки (таких, как `\r`, `\n` или `\n`), вставленных везде, где использовался возврат каретки.

Конечно же, зашифровать получится только те символы, которые встречаются в «Затерянном мире». Программа предупредит вас об исключениях, а затем заменит недостающие символы пробелами.

Для конспирации вам не следует сохранять открытый или шифрованный текст в файле. Лучше воспользуйтесь вырезанием из оболочки и вставкой. Только не забывайте потом копировать в буфер какую-нибудь безобидную информацию, чтобы не оставить уличающих вас следов.

Если вы захотите сделать программу поизящнее, то можете перекопировать текст в буфер прямо из Python с помощью модуля `pyperclip`, написанного Элом Свейгартом. Подробности — на странице https://pypi.org/project/pyperclip/.

Итоги

В этой главе мы поработали с `defaultdic` и `Counter` из модуля `collections`, `choice()` из модуля `random`, а также `replace()`, `enumerate()`, `ord()` и `repr()` из Python Standard Library. В результате у нас получилась программа шифрования, основанная на технике одноразового блокнота, которая производит невзламываемые криптограммы.

Дополнительная литература

Роман «Ключ к Ребекке», написанный Кеном Фоллеттом, по праву стал бестселлером — исторические детали, точные описания Каира во время Второй мировой войны и захватывающая шпионская история не оставляют читателя равнодушным.

Книга «The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography»[1] (Anchor, 2000) Саймона Сингха (Simon Singh) — увлекательный обзор развития криптографии на протяжении нескольких эпох, в том числе и подробное описание применения одноразового блокнота.

Если вам понравилось работать с шифрами, прочитайте книгу «Cracking Codes with Python»[2] (No Starch Press, 2018), написанную Элом Свейгартом. Предназначенная для тех, кто только начинает изучение как криптографии, так и Python, эта книга описывает многие виды шифрования, включая обратное, шифр Цезаря, перестановочный шифр, подстановочный, аффинный и шифр Виженера.

В книге «Impractical Python Projects: Playful Programming Activities to Make You Smarter»[3] (No Starch Press, 2019) Ли Вогана (Lee Vaughan) вы найдете информацию о дополнительных шифрах, таких как Union route cipher (ва-

---

[1] Сингх С. «Книга шифров. Тайная история шифров и их расшифровки».

[2] Свейгарт Э. «Криптография и взлом шифров на Python».

[3] Воган Л. «Непрактичный Python. Занимательные проекты для тех, кто хочет поумнеть».

риация маршрутного шифра), шифре ограждения рельсов и нулевом шифре Треваньона, а также о технике письма с помощью невидимых электронных чернил.

Практический проект: составление графика символов

Если на вашей машине установлена `matplotlib` (инструкции содержатся на с. 31 раздела «Установка библиотек Python»), вы сможете с помощью столбчатой диаграммы визуально представить частотность доступных в романе «Затерянный мир» символов. Это дополнит вывод в оболочке каждого символа и числа его вхождений, реализуемый программой rebecca.py.

Интернет изобилует образцами кода для графиков `matplotlib`, так что достаточно вбить в поиске *«создание простой гистограммы matplotlib»*. Прежде чем строить диаграмму, следует упорядочить количества в порядке уменьшения.

Мнемоника для запоминания наиболее распространенных букв английского языка звучит как «etaoin». Если составить гистограмму в порядке убывания, то можно видеть, что датасет «Затерянного мира» не исключение (рис. 4.2).

Заметьте, что самый распространенный символ — пробел. Это упрощает шифрование пробелов, что еще больше усложняет криптоанализ.

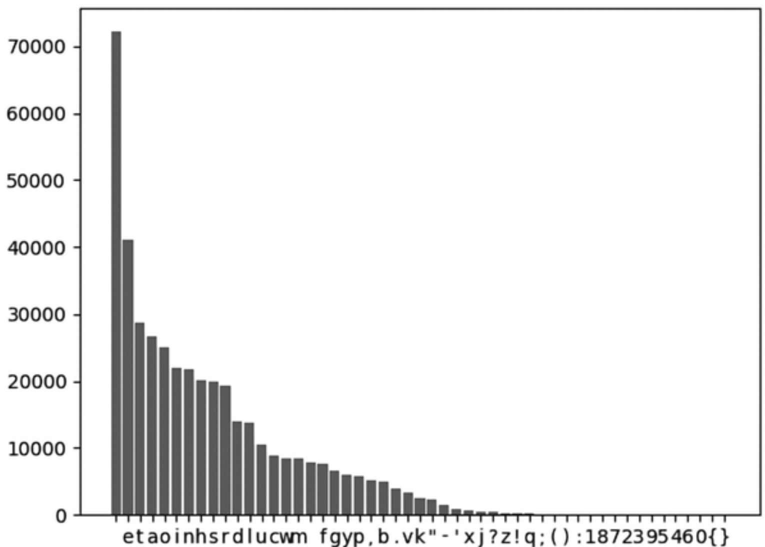


Рис. 4.2. Частотность использования символов в цифровой версии «Затерянного мира»

Решение под названием practice_barchart.py вы найдете в приложении к книге или на ее сайте.

Практический проект: отправка секретов шифром времен Второй мировой войны

Немецкие войска в Северной Африке во время Второй мировой войны действительно использовали роман «Ребекка» в качестве ключа к книжному коду. Вместо кодирования сообщения буква за буквой предложения составлялись из отдельных слов книги, которые обозначались номером страницы, строкой и положением в этой строке.

Скопируйте и отредактируйте программу rebecca.py, чтобы в ней использовались не буквы, а слова. В качестве начальной подсказки я покажу, как с помощью спискового включения загружать текстовый файл в виде списка слов, а не символов:

```
with open('lost.txt') as f:
  words = [word.lower() for line in f for word in line.split()]
  words_no_punct = ["".join(char for char in word if char.isalpha())
               for word in words]

print(words_no_punct[:20]) # Вывести первые 20 слов для проверки
                             работоспособности
```

Вывод должен получиться таким:

```
 ['i', 'have', 'wrought', 'my', 'simple', 'plan', 'if', 'i', 'give', 'one',
'hour', 'of', 'joy', 'to', 'the', 'boy', 'whos', 'half', 'a', 'man']
```

Обратите внимание, что вся пунктуация, включая апострофы, удалена. В сообщениях будет соблюдаться это условие.

Вам также потребуется обработать слова, такие как имена собственные и географические названия, которые не встречаются в «Затерянном мире». Для этого мы применим «режим первой буквы», когда получатель использует только первую букву каждого слова между флагами. В качестве флагов обычно пускают в ход часто встречающиеся слова, например *a* и *the*, в удвоенном виде. Измените их использование, чтобы упростить определение начальных и конечных флагов. В этом случае *a a* указывает начало режима первой буквы, а *the the* — его окончание. К примеру, для обработки фразы *Sidi Muftah with ten tanks* сначала прогоните ее как есть, чтобы определить недостающие слова.

```
Введите открытый или шифрованный текст: sidi muftah with ten tanks
Введите 'encrypt' или 'decrypt': encrypt
Смещение (1-365) = 5
Введите название файла с его расширением: lost.txt
```

```
Слова sidi нет в словаре.
Слова muftah нет в словаре.
Слова tanks нет в словаре.

шифрованный текст =
 [23371, 7491]

дешифрованный открытый текст =
with ten
```

Определив недостающие слова, перефразируйте сообщение, чтобы записать их в режиме первой буквы. В сниппете ниже я выделил первые буквы серым:

```
Введите открытый или шифрованный текст: a a so if do in my under for to all
he the the
with ten a a tell all night kind so the the
Введите 'encrypt' или 'decrypt': encrypt
Смещение (1-365) = 5
Введите название файла с его расширением: lost.txt

шифрованный текст =
[29910, 70641, 30556, 60850, 72292, 32501, 6507, 18593, 41777, 23831, 41833,
16667, 32749, 3350, 46088, 37995, 12535, 30609, 3766, 62585, 46971, 8984,
44083, 43414, 56950]

дешифрованный открытый текст =
a a so if do in my under for to all he the the with ten a a tell all night
kind so the the
```

В «Затерянном мире» символ *a* используется 1864 раза и *the* — 4442 раза. Если придерживаться коротких сообщений, то ключи повторяться не должны. В противном случае вам придется использовать несколько флаговых символов или отключить функцию `check_for_fail()` и согласиться на некоторые повторения.

Можете смело придумать собственный метод обработки проблемных слов. Будучи отличными организаторами, немцы однозначно имели *нечто* на уме, иначе бы изначально не рассматривали использование книжного шифра.

Простое решение этой задачи с использованием метода первой буквы под названием practice_WWII_words.py находится в приложении к книге или на странице https://nostarch.com/real-world-python/.

5

Поиск Плутона

Как говорил Вуди Аллен, 80 % успеха заключается в умении быть на виду. Это определенно объясняет успех Клайда Томбо, необразованного фермерского парня из Канзаса, детство которого пришлось на 1920-е годы. Обладая сильной тягой к астрономии, но не имея при этом денег на колледж, он ткнул пальцем в небо и отправил свои лучшие астрономические наброски в обсерваторию Лоуэлл. К великому удивлению Клайда, обсерватория наняла его в качестве ассистента. А уже год спустя он открыл Плутон и прославился!

Персиваль Лоуэлл, известный астроном и основатель обсерватории Лоуэлла, постулировал присутствие Плутона на основе пертурбаций в орбите Нептуна. Его вычисления оказались ошибочны, но по чистому совпадению он верно спрогнозировал траекторию орбиты Плутона. С 1906 года и до своей кончины в 1916 году Лоуэлл сфотографировал эту планету дважды. Правда, оба раза его команде не удавалось ее разглядеть. Томбо же сфотографировал и распознал Плутон в январе 1930 года, потратив на поиски всего год (рис. 5.1).

Достижение Томбо можно считать экстраординарным. В отсутствие компьютеров использованный им метод был непрактичным, утомительным и капризным. Ночь за ночью ему приходилось неоднократно фотографировать различные участки неба, как правило, из промерзшего купола обсерватории, обдуваемого злыми ветрами. После этого он разбирал и отсеивал все негативы, отыскивая едва заметные следы движения среди густых звездных россыпей.

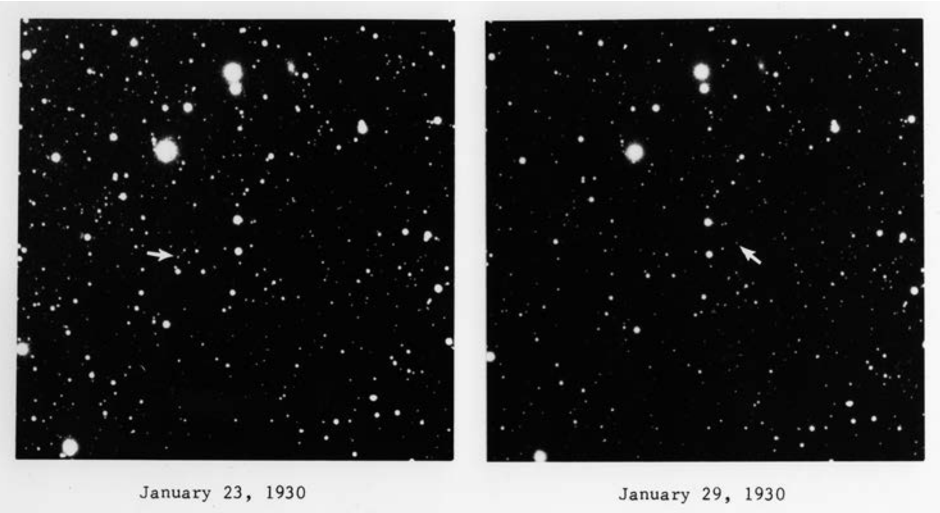


Рис. 5.1. Фотопластины, на которых был обнаружен Плутон (указан стрелкой)

Несмотря на отсутствие компьютера, у него имелось эталонное устройство, называемое *блинк-компаратором*, которое позволяло быстро переключаться между снимками, которые Томбо делал последовательно каждую ночь. При просмотре негативов через это устройство было видно, что звезды остаются на месте, но Плутон, будучи движущимся объектом, высвечивался и затухал, подобно маяку.

В этой главе мы сначала создадим программу Python, повторяющую использованный в начале 1920-х годов блинк-компаратор. Затем переместимся в XXI век и напишем другую программу, которая автоматизирует обнаружение движущихся объектов, используя современные техники компьютерного зрения.

ПРИМЕЧАНИЕ

В 2006 году Международный союз астрономов перевел Плутон в класс планет-карликов. Это было сделано вследствие открытия других близких ему по размеру небесных тел в поясе Койпера, включая Эрис, которое имеет меньший объем, но на 27 % большую массу, чем Плутон.

Проект #7. Воссоздание блинк-компаратора

Плутон можно было фотографировать через телескоп, но нашли его с помощью микроскопа. Блинк-компаратор (рис. 5.2), также называемый

блинк-микроскопом, позволяет устанавливать в него две фотопластины и быстро переключать просмотр с одной на другую. В таком случае любой объект, изменяющий положение на этих двух снимках, будет перескакивать туда-сюда.

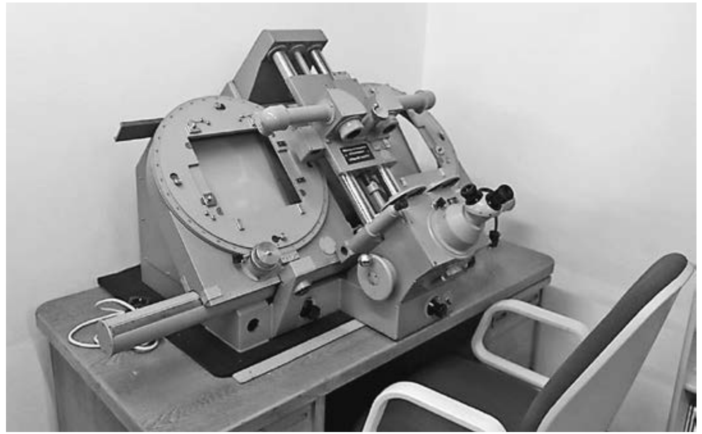


Рис. 5.2. Блинк-компаратор

Чтобы эта техника сработала, фотографии должны быть сделаны с одинаковой экспозицией и в схожих условиях видимости. Самое же главное, чтобы положение звезд на этих снимках идеально совпадало. В бытность Томбо астрономы добивались этого за счет кропотливого ручного труда. Они плавно вели телескоп на протяжении длившихся часами экспозиций, делали снимки, а затем смещали их в блинк-компараторе, добиваясь точного выравнивания. Ввиду столь скрупулезной работы Томбо иногда требовалась целая неделя, чтобы изучить всего пару пластин.

В нашем проекте мы повторим процесс выравнивания пластин и их переключение. При этом мы будем работать с яркими и тусклыми объектами, отметим влияние различия в экспозиции между снимками и сравним использование позитивных изображений и негативных, с которыми работал Томбо.

ЗАДАЧА

Написать программу Python, которая выравнивает два почти идентичных изображения и выводит их по очереди в одном окне с быстрым переключением между изображениями.

Стратегия

Фотографии для проекта уже есть, поэтому нужно лишь их выровнять и реализовать переключение. Выравнивание изображений часто называют *регистрацией*. Этот процесс подразумевает комбинацию вертикальных, горизонтальных или вращательных преобразований одного из изображений. Если вам доводилось делать панорамный снимок с помощью цифровой камеры, то вы видели регистрацию в действии.

Состоит она из следующих этапов.

1. Обнаруживаются отличительные признаки в каждом изображении.
2. Выполняется числовое описание каждого признака.
3. Полученные числовые дескрипторы используются для сопоставления идентичных признаков в каждом изображении.
4. Одно изображение следует деформировать так, чтобы совпадающие объекты имели одинаковое расположение пикселей.

Чтобы все это сработало как надо, изображения должны быть одного размера и охватывать приблизительно одну область.

К счастью, пакет OpenCV для Python содержит алгоритмы, которые выполняют все эти действия. Если вы пропустили главу 1, то стоит почитать об OpenCV на с. 31.

После регистрации изображений отобразите их в одном окне, добившись точного наложения, а затем выполните их поочередный показ заданное количество раз. Это можно также реализовать с помощью OpenCV.

Данные

Нужные изображения находятся в каталоге Chapter_5 файлов книги, которые можно скачать с https://nostarch.com/real-world-python/. Структура каталогов должна выглядеть, как показано на рис. 5.3. Выполнив скачивание, не меняйте эту структуру, равно как содержимое каталогов и их имена.

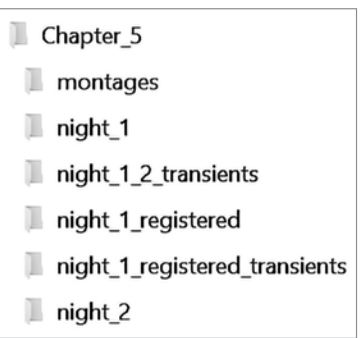


Рис. 5.3. Структура каталогов для проекта 7

Каталоги night_1 и night_2 содержат входные изображения, с которыми мы будем работать. В теории это должны быть снимки одного и того же участка звездного неба, сделанные в разные ночи. Здесь же у нас два одинаковых снимка

ночного неба, на которые я добавил искусственный *транзиент*. Транзиент — это небесный объект, чье движение можно обнаружить за относительно короткие промежутки времени. К подобным объектам относятся кометы, астероиды и планеты, поскольку их движение можно видеть на более статичном фоне галактики.

В табл. 5.1 кратко описывается содержимое каталога night_1. В нем находятся файлы, в названиях которых присутствует слово *left*. Это означает, что их следует размещать с левой стороны блинк-компаратора. Изображения в каталоге night_2 содержат в своих названиях слово *right* и должны размещаться с правой стороны.

Таблица 5.1. Файлы в каталоге night_1

Имя файла	**Описание**
1_bright_transient_left.png	Содержит большой, яркий транзиент
2_dim_transient_left.png	Содержит тусклый транзиент диаметром 1 пиксель
3_diff_exposures_left.png	Содержит тусклый транзиент с переэкспонированным фоном
4_single_transient_left.png	Содержит яркий транзиент только в левом изображении
5_no_transient_left.png	Звездное поле без транзиентов
6_bright_transient_neg_left.png	Негатив первого файла, показывающий тип использованного Томбо изображения

Рисунок 5.4 представляет пример одного из изображений. Стрелка указывает на транзиент (но она не является частью файла изображения).

Чтобы воссоздать сложность идеального выравнивания телескопа из ночи в ночь, я немного сместил изображения в каталоге night_2 относительно изображений из каталога night_1. Нам потребуется перебрать содержимое этих каталогов, регистрируя и сравнивая каждую пару снимков. По этой причине в каждом каталоге должно находиться одинаковое число файлов, а их имена должны гарантировать правильное сопоставление пар снимков.

Код блинк-компаратора

Приведенный ниже код `blink-comparator.py` в цифровом виде повторяет блинк-компаратор. Найдите эту программу в каталоге Chapter_5 на сайте. Вам также понадобятся каталоги, описанные в предыдущем разделе. Код находится в директории на уровень выше каталогов night_1 и night_2.

Рис. 5.4. 1_bright_transient_left.png со стрелкой, указывающей на транзиент

Импорт модулей и присваивание константы

Код листинга 5.1 импортирует необходимые для выполнения программы модули и определяет константу, которой присваивается минимальное число совпадений ключевых точек. *Ключевые точки*, также называемые точками интереса, содержат признаки изображения, которые можно использовать для его описания. Они обычно характеризуются резкими изменениями в интенсивности, например углами или, как в данном случае, звездами.

Листинг 5.1. Импорт модулей и присваивание константе количества совпадений ключевых точек

blink_comparator.py, часть 1

```
import os
from pathlib import Path
import numpy as np
import cv2 as cv

MIN_NUM_KEYPOINT_MATCHES = 50
```

Вначале выполняется импорт модуля операционной системы, который будет использоваться для перечисления содержимого каталогов. Далее — импорт `pathlib`, вспомогательного модуля, упрощающего работу с файлами и каталогами. Завершается импорт библиотеками `NumPy` и `cv` (OpenCV) для работы с изображениями. Если вы пропустили главу 1, то инструкции по установке `NumPy` вы найдете на с. 33.

В конце присваиваем постоянную переменную, определяющую минимальное число совпадений ключевых точек. В идеале нужно добиться использования

наименьшего значения, которое будет давать приемлемый результат регистрации. В данном же проекте алгоритм выполняется настолько быстро, что это значение можно увеличить без существенной потери производительности.

Определение функции main()

В листинге 5.2 мы определяем первую часть функции `main()`, отвечающей за запуск программы. Сначала создаются списки и пути каталогов, используемые для обращения к различным файлам изображений.

Листинг 5.2. Определение первой части функции main(), используемой для управления файлами и каталогами

blink_comparator.py, часть 2

```
def main():
    """Перебираем 2 папки с парными изображениями, делаем регистрацию
    и просматриваем изображения на блинк-компараторе."""
    night1_files = sorted(os.listdir('night_1'))
    night2_files = sorted(os.listdir('night_2'))
    path1 = Path.cwd() / 'night_1'
    path2 = Path.cwd() / 'night_2'
    path3 = Path.cwd() / 'night_1_registered'
```

Начинаем с определения `main()` и затем используем метод `listdir()` модуля `os` для создания списка имен файлов в каталогах night_1 и night_2. Для night_1 `listdir()` возвращает следующее:

```
['1_bright_transient_left.png', '2_dim_transient_left.png',
'3_diff_exposures_
left.png', '4_no_transient_left.png', '5_bright_transient_neg_left.png']
```

Обратите внимание, что `os.listdir()` не устанавливает порядок для файлов по их возвращении. Порядок определяется операционной системой, и это означает, что в macOS он будет отличаться от порядка в Windows. Чтобы обеспечить согласованность списков и пар файлов, обертываем `os.listdir()` встроенной функцией `sorted()`. Эта функция вернет файлы в числовом порядке на основе первого символа их имен.

Далее присваиваем имена путей переменным с помощью класса `Path` модуля `pathlib`. Первые две переменные указывают на два входных каталога, а третья — на выходной каталог, где будут находиться выровненные изображения.

Модуль `pathlib`, появившийся в Python 3.4, является альтернативой `os.path` для обработки путей файлов. Модуль `os` рассматривает пути как строки, что может быть громоздко и требует использования функциональности из разных частей стандартной библиотеки. Вместо этого модуль `pathlib` трактует пути как объекты и собирает необходимую функциональность в одном месте. Официальная

документация для `pathlib` находится на странице https://docs.python.org/3/library/pathlib.html.

Для первой части пути каталога используем метод класса `cwd()`, чтобы получить текущую рабочую директорию. Если у вас есть хотя бы один объект `Path`, вы можете использовать в обозначении пути объекты и строки. Строки, представляющие имя каталога, можно объединять через символ `/`. Если вы знакомы с модулем `os`, то это аналогично использованию `os.path.join()`.

Обратите внимание, что вам потребуется выполнять программу из каталога проекта. При вызове из другого места файловой системы она даст сбой.

Выполнение цикла в main()

Листинг 5.3 продолжает функцию `main()`, выполняя программу в виде большого цикла `for`. Этот цикл будет получать по одному файлу из каждого каталога night, загружать их в виде изображения в оттенках серого, находить на каждом изображении совпадающие ключевые точки, использовать эти точки для деформации (или *регистрации*) первого изображения, подгоняя его под соответствие со вторым, затем сохранять зарегистрированное первое изображение и сравнивать его с оригинальным вторым. Я также включил несколько дополнительных действий для контроля качества, которые вы можете закомментировать, когда получите удовлетворительные результаты.

Листинг 5.3. Выполнение цикла программы в main()

blink_comparator.py, часть 3

```
for i, _ in enumerate(night1_files):
    img1 = cv.imread(str(path1 / night1_files[i]), cv.IMREAD_GRAYSCALE)
    img2 = cv.imread(str(path2 / night2_files[i]), cv.IMREAD_GRAYSCALE)
    print("Comparing {} to {}.\n".format(night1_files[i],
           night2_files[i]))
  ❶ kp1, kp2, best_matches = find_best_matches(img1, img2)
    img_match = cv.drawMatches(img1, kp1, img2, kp2,
                               best_matches, outImg=None)
    height, width = img1.shape
    cv.line(img_match, (width, 0), (width, height), (255, 255, 255), 1)
  ❷ QC_best_matches(img_match)  # Закомментировать для игнорирования
                                  программой
    img1_registered = register_image(img1, img2, kp1, kp2, best_matches)

  ❸ blink(img1, img1_registered, 'Check Registration', num_loops=5)
    out_filename = '{}_registered.png'.format(night1_files[i][:-4])
    cv.imwrite(str(path3 / out_filename), img1_registered) # Файл будет
                                                             переписан!
    cv.destroyAllWindows()
    blink(img1_registered, img2, 'Blink Comparator', num_loops=15)
```

Цикл начинается с перечисления списка `night1_files`. Встроенная функция `enumerate()` добавляет каждому его элементу порядковый номер и возвращает этот номер вместе с самим элементом. Поскольку нам понадобится только номер, мы используем для элемента списка одиночное нижнее подчеркивание (_). По соглашению одиночное подчеркивание указывает на временную или незначительную переменную. Оно также избавляет от лишних действий программы по проверке кода, выполняемых Pylint. Если бы мы использовали здесь имя переменной вроде `infile`, Pylint бы «ругалась» сообщением *unused variable* (неиспользованная переменная).

```
W: 17,11: Unused variable 'infile' (unused-variable)
```

Далее с помощью OpenCV загружаем изображение вместе с его парой из списка `night2_files`. Обратите внимание, что для метода `imread()` нужно преобразовать путь в строку. Нам также потребуется перевести изображение в оттенки серого. Это позволит работать всего с одним каналом, представляющим интенсивность. Для отслеживания действий, происходящих в цикле, мы выводим сообщение, указывающее, какие файлы в данный момент сравниваются.

Теперь находим ключевые точки и их лучшие совпадения ❶. Функция `find_best_matches()`, которую мы определим чуть позже, будет возвращать эти значения в виде трех переменных: `kp1` и `kp2`, которые представляют ключевые точки первого и второго загруженных изображений, и `best_matches`, представляющей список совпавших ключевых точек.

Так мы сможем визуально проверить совпадения и отразить их на `img1` и `img2` при помощи метода OpenCV `drawMatches()`. В качестве аргументов этот метод получает каждое изображение с его ключевыми точками, список точек с наиболее точным совпадением и выходное изображение. В этом случае аргумент выходного изображения установлен как `None`, поскольку нам потребуется только рассмотреть вывод, не сохраняя его в файл.

Для различения двух изображений рисуем вертикальную белую линию вдоль правого края `img1`. Сначала получаем высоту и ширину изображения, используя `shape`. Далее вызываем метод OpenCV() `line()` и передаем ему изображение, на котором будем рисовать координаты начала и конца линии, а также ее цвет и толщину. Обратите внимание, что это цветное фото, поэтому для представления белого цвета потребуется полный кортеж BGR `(255, 255, 255)`, а не одно значение интенсивности (255), используемое в полутоновых изображениях с оттенками серого.

Теперь вызываем функцию контроля качества — которую определим позже — для отображения совпадений ❷. На рис. 5.5 показан пример вывода. По желанию вы можете закомментировать эту строку после того, как убедитесь в корректном выполнении программы.

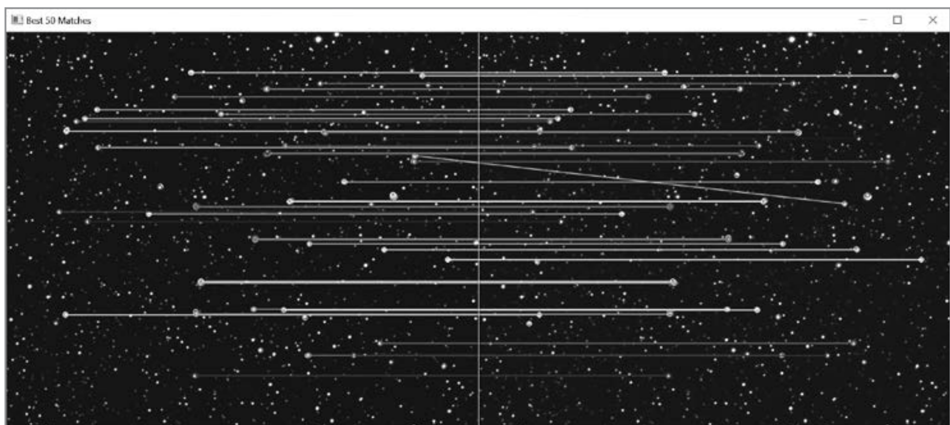


Рис. 5.5. Пример вывода функции QC_best_matches()

После обнаружения и проверки наилучших совпадений ключевых точек переходим к регистрации первого изображения относительно второго. Для этого используем функцию, которая также будет написана позднее. Ей передаются два изображения, их ключевые точки и список лучших совпадений.

Блинк-компаратор под названием `blink()` — это еще одна функция, которую мы создадим позднее. Здесь мы ее вызываем, чтобы увидеть эффект от регистрации на первом изображении. Передаем этой функции оригинальное и зарегистрированное изображения, имя для экранного окна и количество переключений между изображениями (бл*и*нков), которое хотим выполнить ❸. Функция будет переключаться между двумя изображениями. Величина «дрожания» изображения, которое вы увидите, будет зависеть от величины деформации, потребовавшейся для сопоставления с `img2`. Это еще одна строка, которую можно закомментировать, убедившись в правильной работе программы.

Далее сохраняем зарегистрированное изображение в каталоге night_1_registered, на который указывает переменная `path3`. Начинаем с присваивания имени файла переменной, ссылающейся на исходное имя файла, добавив в конце `_registered.png`. Чтобы не повторять расширение файла в его имени, используем срез по индексу (`[:-4]`) для его удаления перед добавлением нового окончания. В завершение с помощью `imwrite()` сохраняем файл. Обратите внимание, что так мы перезаписываем существующие файлы с тем же именем, не получая предупреждения.

При поиске транзиентов нам нужно четкое изображение, поэтому мы вызываем метод для закрытия текущих окон OpenCV. Затем еще раз вызываем функцию `blink()`, передавая ей зарегистрированное изображение, второе изображение,

имя окна и количество переключений изображения. Первые изображения показаны рядом на рис. 5.6. Сможете найти транзиент?

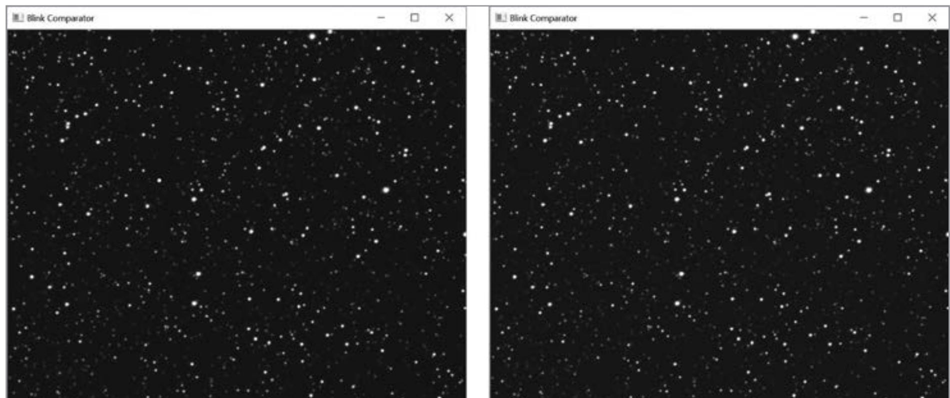


Рис. 5.6. Окна блинк-компаратора для первого изображения в каталогах night_1_registered и night_2

Поиск наилучших совпадений ключевых точек

Пришло время прописать функции, используемые в `main()`. Листинг 5.4 определяет функцию, которая находит наилучшие совпадения ключевых точек между каждой парой изображений, взятых из каталогов night_1 и night_2. Она должна находить, описывать и сопоставлять ключевые точки, генерировать список их совпадений, а затем сокращать этот список согласно константе, указывающей минимальное количество приемлемых ключевых точек. Эта функция возвращает список ключевых точек для каждого изображения и список наиболее точных совпадений.

Листинг 5.4. Определение функции для нахождения лучших совпадений ключевых точек

blink_comparator.py, часть 4

```
def find_best_matches(img1, img2):
    """Вернуть список ключевых точек и список наилучших совпадений
    для двух изображений."""
    orb = cv.ORB_create(nfeatures=100) # Инициировать объект ORB.
  ❶ kp1, desc1 = orb.detectAndCompute(img1, mask=None)
    kp2, desc2 = orb.detectAndCompute(img2, mask=None)
    bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
  ❷ matches = bf.match(desc1, desc2)
    matches = sorted(matches, key=lambda x: x.distance)
    best_matches = matches[:MIN_NUM_KEYPOINT_MATCHES]

return kp1, kp2, best_matches
```

Начинаем с определения функции, которая получает в качестве аргументов два изображения. Функция `main()`берет эти изображения из входных каталогов при каждом выполнении цикла `for`.

Далее с помощью метода OpenCV `ORB_create()` создаем объект `orb`. ORB — это акроним от *O*riented FAST and *R*otated *B*RIEF (ориентированные FAST и повернутые BRIEF). FAST — сокращение от *F*eatures from *A*ccelerated *S*egment *T*est (признаки из ускоренной проверки сегмента). Это быстрый, эффективный и бесплатный алгоритм для *обнаружения* ключевых точек. А чтобы *описать* ключевые точки таким способом, который позволит сравнить их по нескольким изображениям, потребуется BRIEF, то есть *B*inary *R*obust *I*ndependent *E*lementary *F*eatures (двоичные устойчивые независимые элементарные признаки). Этот алгоритм также быстр, компактен и имеет открытый исходный код.

ORB совмещает FAST и BRIEF в алгоритм совпадения, который обнаруживает области на изображениях, где значения пикселей изменяются резко, после чего записывает позиции этих различающихся областей как *ключевые точки*. Затем с помощью числовых массивов, или *дескрипторов,* ORB описывает признак, обнаруженный в ключевой точке, определяя вокруг этой точки небольшую область, называемую *патчем*.

Внутри патча изображения алгоритм использует шаблонный метод, чтобы получить типичные образцы интенсивности. Затем он сравнивает предварительно выбранные пары образцов и преобразует их в двоичные строки — *вектора признаков* (рис. 5.7).

Вектор — это последовательность чисел. *Матрица* — прямоугольный массив чисел, расположенных в строках и столбцах, которые рассматриваются как единая сущность и управляются согласно установленным правилам. *Вектор признаков* — матрица с одной строкой и несколькими столбцами. Для его создания алгоритм преобразует пары образцов в двоичные последовательности, присоединяя в конец вектора 1, если первый образец имеет наибольшую интенсивность, и 0, если верно обратное.

Ниже показаны примеры векторов признаков. Я сократил список векторов, потому что ORB обычно сравнивает и записывает 512 пар образцов.

```
V1 = [010010110100101101100--snip--]
V2 = [100111100110010101101--snip--]
V3 = [001101100011011101001--snip--]
--snip--
```

Эти дескрипторы выступают как цифровые отпечатки признаков. Для компенсации вращения и изменения масштаба OpenCV использует дополнительный код. Это позволяет ей сопоставлять схожие признаки, даже если их размеры и ориентация отличаются (рис. 5.8).

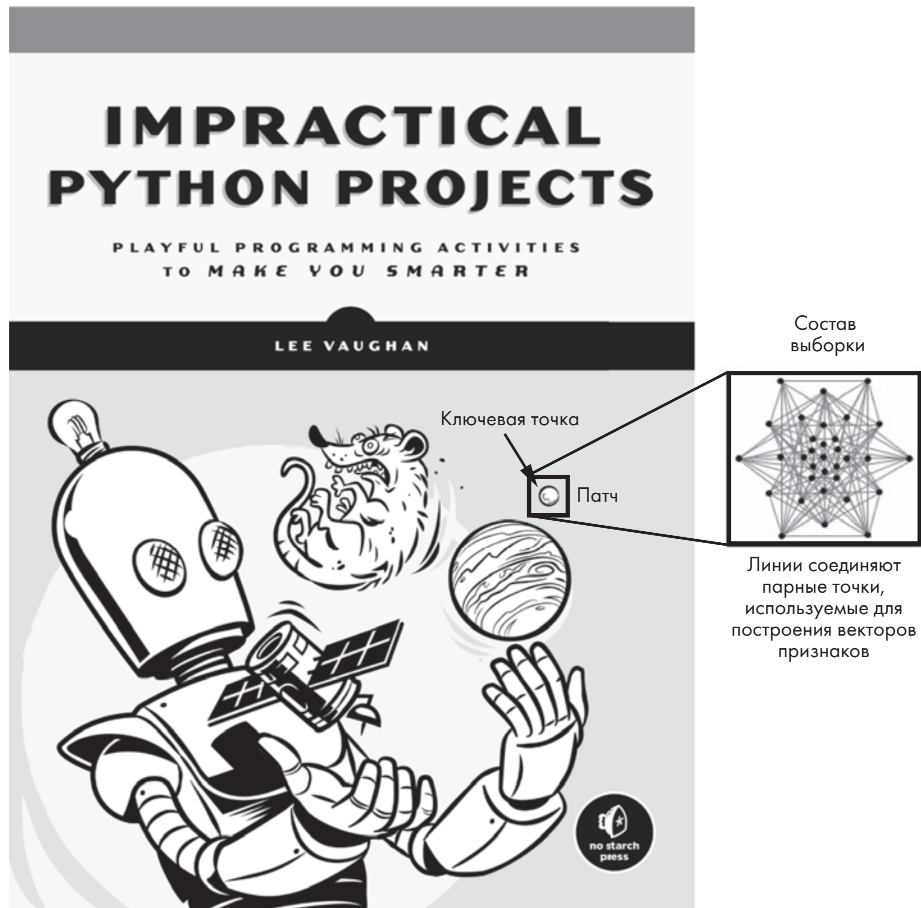


Рис. 5.7. Мультяшный пример генерации дескрипторов ключевых точек

При создании объекта ORB мы можем указать количество ключевых точек для оценки. По умолчанию этот метод рассматривает 500 ключевых точек, но для регистрации изображений в нашем проекте вполне хватит и 100.

Далее с помощью метода `orb.detectAndCompute()` ❶ находим ключевые точки и их дескрипторы. Передаем ему `img1`, после чего повторяем код для `img2`.

После обнаружения и описания ключевых точек необходимо отыскать среди них общие для обоих изображений. Прежде всего создадим объект `BFMatcher` для измерения расстояния. Этот объект берет дескриптор одного признака первого изображения и с помощью расстояния Хэмминга сравнивает его со всеми признаками второго, в результате возвращая самый близкий.

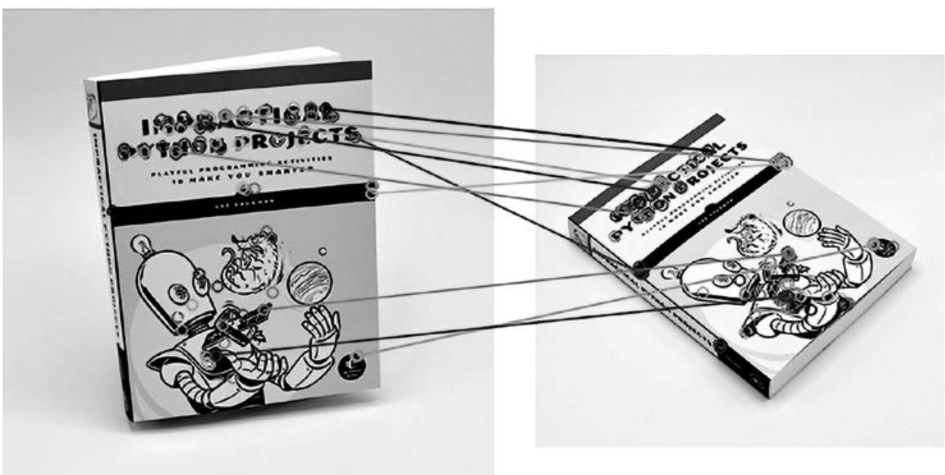


Рис. 5.8. OpenCV может сопоставлять ключевые точки, несмотря на разницу в их масштабе и ориентации

Для двух строк одинаковой длины *расстояние Хэмминга* представляет количество позиций, или индексов, в которых соответствующие значения отличаются. Для следующих векторов признаков несовпадающие позиции показаны жирным шрифтом, а расстояние Хэмминга равно 3:

```
1001011001010
1100111001010
```

Переменная `bf` представляет объект `BFMatch`. Вызываем метод `match()` и передаем ему дескрипторы для двух изображений ❷. Возвращаемый список объектов `Dmatch` присваиваем переменной `matches`.

Для самых точных совпадений расстояние Хэмминга оказывается наименьшим, поэтому упорядочиваем эти объекты по возрастанию, чтобы переместить их к началу списка. Обратите внимание, что мы используем лямбда-функцию вместе с атрибутом объекта `distance`. *Лямбда-функция* — это небольшая одноразовая безымянная функция, определяемая на лету. Слова и символы, которые следуют сразу за `lambda`, являются параметрами. Выражения перечислены после двоеточия, а возвращения происходят автоматически.

Поскольку нам требуется минимальное число совпадений ключевых точек, установленное в начале программы, мы урезаем список `matches`, создавая его укороченную версию. Наилучшие совпадения указаны в начале, поэтому срез делаем от начала `matches` и до значения, указанного в `MIN_NUM_KEYPOINT_MATCHES`.

В этот момент мы все еще работаем с магическими объектами:

```
best matches = [<DMatch 0000028BEBAFBFB0>, <DMatch 0000028BEBB21090>, --snip--
```

К счастью, OpenCV знает, как их обработать. Завершаем функцию возвращением двух множеств ключевых точек и списка максимально точно совпавших объектов.

Проверка лучших совпадений

В листинге 5.5 определяется короткая функция, которая позволит визуально проверить соответствие ключевых точек. Результаты этой функции вы видели на рис. 5.5. Путем инкапсулирования этих задач в функцию можно уменьшить беспорядок в `main()` и дать пользователю возможность отключить эту функциональность, закомментировав всего одну строку.

Листинг 5.5. Определение функции для проверки лучших совпадений ключевых точек

blink_comparator.py, часть 5

```
def QC_best_matches(img_match):
    """Рисуем наилучшие совпадения ключевых точек, соединенные
    цветными линиями."""
    cv.imshow('Best {} Matches'.format(MIN_NUM_KEYPOINT_MATCHES), img_match)
    cv.waitKey(2500) # Keeps window active 2.5 seconds.
```

Определяем функцию с одним параметром: сопоставляемым изображением. Это изображение было сгенерировано функцией `main()` в листинге 5.3. Оно состоит из левого и правого изображений, где ключевые точки нарисованы в виде цветных кругов, а цветные линии соединяют их связанные пары.

Далее выполняется вызов метода OpenCV `imshow()` для отображения окна. При именовании окна можно использовать метод `format()`. Передаем ему константу, указывая тем самым минимальное количество совпадений ключевых точек.

Завершаем функцию, давая пользователю 2.5 секунды для просмотра окна. Обратите внимание, что метод `waitKey()` не закрывает это окно. Он просто приостанавливает программу на заданное время. По истечении периода ожидания появляются новые окна и программа возобновляет выполнение.

Регистрация изображений

В листинге 5.6 определена функция для регистрации первого изображения относительно второго.

Листинг 5.6. Определение функции для регистрации одного изображения относительно другого

blink_comparator.py, часть 6

```
def register_image(img1, img2, kp1, kp2, best_matches):
    """Вернем первое изображение, зарегистрированное по второму
    изображению."""
    if len(best_matches) >= MIN_NUM_KEYPOINT_MATCHES:
        src_pts = np.zeros((len(best_matches), 2), dtype=np.float32)
        dst_pts = np.zeros((len(best_matches), 2), dtype=np.float32)
      ❶ for i, match in enumerate(best_matches):
            src_pts[i, :] = kp1[match.queryIdx].pt
            dst_pts[i, :] = kp2[match.trainIdx].pt
        h_array, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC)
      ❷ height, width = img2.shape # Получим размеры изображения 2.
        img1_warped = cv.warpPerspective(img1, h_array, (width, height))
        return img1_warped

    else:
        print("WARNING: Number of keypoint matches < {}\n".format
              (MIN_NUM_KEYPOINT_MATCHES))
        return img1
```

Определяем функцию, получающую в качестве аргументов два входных изображения, списки их ключевых точек и список объектов `DMatch`, возвращенный функцией `find_best_matches()`. Далее загружаем в массивы `NumPy` расположение лучших совпадений. Начинаем с условной конструкции, которая проверяет, что длина списка лучших совпадений равна или превышает константу `MIN_NUM_KEYPOINT_MATCHES`. Если да, то инициализируем два массива `NumPy` с количеством строк, соответствующим числу лучших совпадений.

Метод `np.zeros()` библиотеки `NumPy` возвращает новый массив заданной формы и типа данных, заполненный нулями. К примеру, следующий фрагмент производит заполненный нулями массив в три строки высотой и два столбца шириной:

```
>>> import numpy as np
>>> ndarray = np.zeros((3, 2), dtype=np.float32)
>>> ndarray
array([[0., 0.],
       [0., 0.],
       [0., 0.]], dtype=float32)
```

В реальном коде размеры массивов будут не менее 50 × 2, поскольку мы обозначили не менее 50 совпадений.

Теперь нумеруем список `matches` и начинаем заполнять массивы фактическими данными ❶. Для исходных точек используем атрибут `queryIdx.pt`, чтобы получить для `kp1` индекс дескриптора в списке дескрипторов. Повторяем этот процесс для следующего множества точек, но используем уже атрибут `trainIdx.pt`.

Терминология `query/train` несколько путает, но, по сути, относится к первому и второму изображениям соответственно.

Теперь применим *гомографию* — трансформацию с использованием матрицы 3 × 3, которая сопоставляет точки на одном изображении с соответствующими им точками на другом изображении. Два изображения могут быть связаны гомографией, если они оба отображают одну и ту же плоскость под разными углами или если они сделаны одной камерой, повернутой вокруг своей оптической оси без смещения. Для правильного выполнения гомографии требуется не менее четырех соответствующих точек на двух изображениях.

Эта техника подразумевает, что сопоставляемые точки реально соответствуют друг другу. Но при внимательном рассмотрении рис. 5.5 и 5.8 можно заметить, что признаки сопоставлены не идеально. На рис. 5.8 около 30 % совпадений ошибочны.

К счастью, в OpenCV есть метод `findHomography()` для обнаружения выбросов, который называется *консенсусом случайной выборки* (RANSAC, random sample consensus). RANSAC получает случайные выборки совпадающих точек, находит математическую модель, объясняющую их распределение, и отдает предпочтение той модели, которая прогнозирует их наибольшее число. После этого выбросы исключаются. Рассмотрим, к примеру, точки в рамке «Сырые данные» на рис. 5.9.

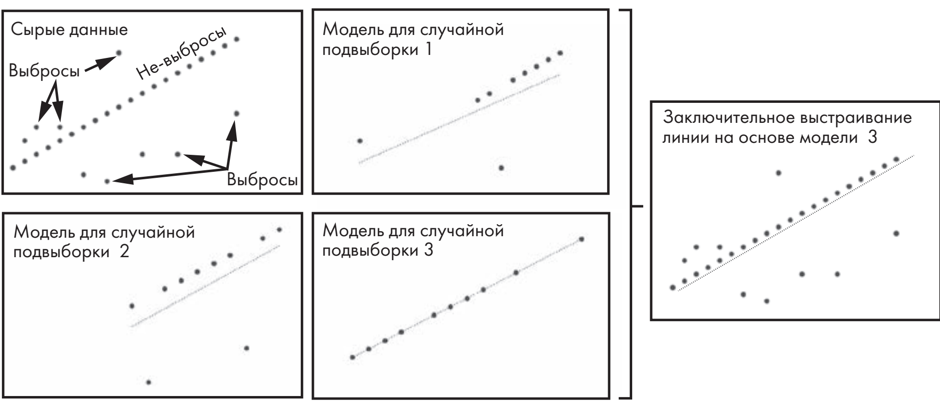


Рис. 5.9. Пример выстраивания линии с использованием RANSAC для игнорирования выбросов

Итак, нам нужно провести линию через истинные точки данных (называемые *не-выбросы*) и проигнорировать меньшее число ложных точек (*выбросов*). С помощью RANSAC мы случайно отбираем подмножество исходных точек данных, подгоняем под них линию, а затем повторяем процесс заданное число

раз. Каждое уравнение аппроксимации прямой впоследствии применяется ко всем точкам. Прямая, которая проходит через большее число точек, используется для заключительной аппроксимации. На рис. 5.9 это показано в самой правой рамке.

Для выполнения `findHomography()` передаем ей исходную и целевую точки и вызываем метод RANSAC. В ответ получаем массив `NumPy` и маску. Маска указывает точки не-выбросов и выбросов, иначе говоря, удачные совпадения и неудачные. Ее можно использовать для того, чтобы рисовать, например, только хорошие совпадения.

В заключение деформируем первое изображение так, чтобы оно идеально сопоставлялось со вторым. Для этого нам потребуются измерения второго изображения, поэтому используем `shape()`, чтобы получить высоту и ширину `img2` ❷. Передаем эту информацию вместе с `img1` и массивом гомографии `h_array` методу `warpPerspective()`. Возвращаем зарегистрированное изображение, которое представлено массивом `NumPy`.

Если количество совпадений ключевых точек меньше минимального числа, установленного в начале программы, изображение может быть выровнено *неточно*. Поэтому выводим предупреждение и возвращаем оригинальное незарегистрированное изображение. Это позволит функции `main()` продолжить перебирать изображения в каталогах без перерыва. Если регистрация пройдет плохо, пользователь окажется в курсе неполадки, так как пара проблемных изображений не будет точно выровнена в окне блинк-компаратора. При этом в оболочке появится сообщение об ошибке.

```
Сравнить 2_dim_transient_left.png to 2_dim_transient_right.png.
ПРЕДУПРЕЖДЕНИЕ: Количество совпадений ключевых точек < 50
```

Создание блинк-компаратора

Код листинга 5.7 определяет функцию для запуска блинк-компаратора, после чего вызывает `main()`, если программа выполняется в автономном режиме. Функция `blink()` перебирает заданный диапазон, в одном окне показывая сначала зарегистрированное изображение, а за ним второе. Каждое изображение она показывает только в течение 1/3 с. Именно такую частоту использовал Клайд Томбо в своем блинк-компараторе.

Листинг 5.7. Определение функции для повторяющейся смены изображений

blink_comparator.py, часть 7

```
def blink(image_1, image_2, window_name, num_loops):
    """Копия блинк-компаратора с двумя изображениями"""
    for _ in range(num_loops):
```

```
        cv.imshow(window_name, image_1)
        cv.waitKey(330)
        cv.imshow(window_name, image_2)
        cv.waitKey(330)

if __name__ == '__main__':
    main()
```

Определяем функцию `blink()` с четырьмя параметрами: двумя файлами изображений, именем окна и количеством выполняемых переключений. Цикл `for` начинается с диапазона, установленного на количество переключений изображений. Поскольку доступ к меняющемуся индексу нам не требуется, мы применяем одиночное нижнее подчеркивание, указывая на использование незначительной переменной. Как уже говорилось, это предотвратит выдачу проверяющими код программами предупреждения «unused variable».

Теперь вызываем метод OpenCV `imshow()`, передавая ему имя окна и первое изображение. Это будет *зарегистрированное* первое изображение. Затем приостанавливаем программу на 330 мс, продолжительность, рекомендованную Клайдом Томбо.

Повторяем две предыдущие строки кода для второго изображения. Поскольку эти два изображения выровнены, единственное, что изменится в окне, — это транзиенты. Если транзиент содержится только на одном изображении, то он будет мерцать. Если же транзиент находится на обоих снимках, то он будет смещаться туда-сюда.

Завершаем программу отдельным кодом, который позволит ей выполняться в автономном режиме или быть импортированной в качестве модуля.

Использование блинк-компаратора

Прежде чем запускать blink_comparator.py, приглушите освещение в комнате, чтобы сымитировать просмотр изображений через окуляры устройства. Теперь запускайте программу. Сначала вы должны увидеть две отчетливые яркие точки, мерцающие возле центра изображения. На следующей паре изображений те же точки станут очень маленькими — всего пиксель в диаметре, но вы все равно должны их разглядеть.

Третий цикл покажет тот же мелкий транзиент, но на этот раз общая яркость второго изображения будет больше, чем первого. Вы все равно должны обнаружить транзиент, хотя это будет уже намного сложнее. Именно поэтому Томбо приходилось с осторожностью делать и обрабатывать снимки с согласованной экспозицией.

Четвертый цикл содержит один транзиент, представленный на левом изображении. Теперь он должен уже не скакать туда-сюда, как на предыдущих снимках, а мерцать.

Пятая пара изображений представляет контрольные снимки без транзиентов. Именно такую картину астрономы и наблюдают в большинстве случаев: досадно неподвижные звездные россыпи.

Заключительный цикл использует негативные варианты первой пары изображений. Яркий транзиент здесь проявляется в виде мигающих черных точек. Именно такой тип изображения использовал Клайд Томбо, поскольку это экономило время. Ввиду того что черную точку так же легко обнаружить, как и белую, он не видел необходимости распечатывать позитивные изображения для каждого негатива.

Вдоль левой стороны зарегистрированного негатива видна черная полоса, которая показывает размер смещения, необходимый для выравнивания изображений (рис. 5.10). Ее не удастся увидеть на позитивных снимках, потому что она сливается с черным фоном.

Рис. 5.10. Негативное изображение, 6_bright_transient_neg_left_registered.png

Во всех циклах можно заметить тусклую звезду, мерцающую в верхнем левом углу каждой пары снимков. Это не транзиент, а ложноположительный результат, вызванный *краевым эффектом*. Краевой эффект — изменение в изображении вследствие его неверного выравнивания. Опытный астроном проигнорировал бы эту тусклую звезду, потому что она появляется очень близко к краю изображения и возможный транзиент не перемещается между изображениями, а просто тускнеет.

Причина появления этого ложноположительного эффекта иллюстрируется на рис. 5.11. Так как в первый кадр попадает только часть звезды, ее яркость меньше, чем на втором изображении.

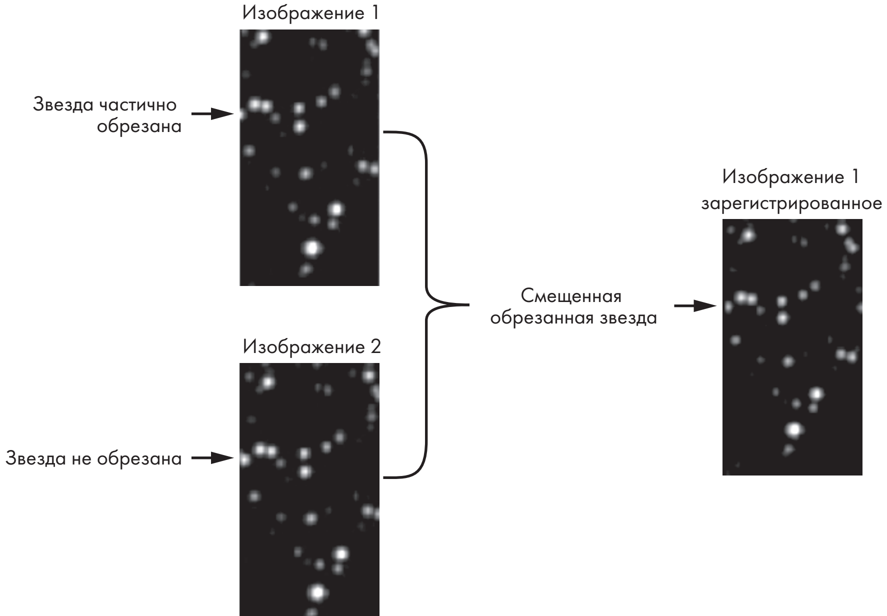


Рис. 5.11. Звезда на изображении 1 обрезана и поэтому выглядит более тусклой, чем на изображении 2

Люди могут распознавать краевые эффекты краев интуитивно, но компьютерам для этого нужны явные правила. В следующем проекте мы решим эту проблему, исключив при поиске транзиентов края изображения.

Проект #8. Обнаружение астрономических транзиентов путем дифференцирования изображений

Блинк-компараторы, которые некогда были столь же значимы, как сами телескопы, теперь пылятся в музеях. Астрономы в них больше не нуждаются, поскольку современные техники дифференцирования изображений намного лучше обнаруживают движущиеся объекты, чем человеческий глаз. Сегодня каждый этап проделанной Клайдом Томбом работы можно выполнить с помощью компьютеров.

Давайте вообразим, что мы интерны, проходящие практику в обсерватории. Наша задача — создать цифровой рабочий процесс для астронома прошлого века, который никак не может оторваться от своего ржавеющего блинк-компаратора.

ЗАДАЧА

Написать программу Python, которая получает два зарегистрированных изображения и выделяет любые отличия между ними.

Стратегия

Вместо алгоритма, сменяющего изображения, теперь нам нужен такой, который будет автоматически находить транзиенты. В этом процессе по-прежнему потребуются зарегистрированные изображения, но для удобства мы будем использовать всего одно, уже полученное в проекте 7.

Обнаружение отличий между изображениями — весьма стандартная возможность, которую OpenCV предоставляет вместе с методом абсолютных разниц `absdiff()`, предназначенным именно для такой цели. Этот метод поэлементно получает различия в двух массивах. Но просто обнаружить различия недостаточно. Программа должна также распознавать, что отличия имеются, и показывать пользователю только изображения, содержащие транзиенты. Ведь астрономам и без того есть чем заняться, например понизить какие-нибудь планеты в статусе.

Поскольку интересующие нас объекты располагаются на черном фоне, а совпадающие яркие объекты удаляются, любой оставшийся яркий объект заслуживает внимания. А так как шанс одновременно обнаружить среди звездной россыпи более одного транзиента весьма мал, то обнаружение одного или двух отличий достаточно для привлечения внимания астронома.

Код для детектора транзиентов

Приведенный ниже код transient_detector.py автоматизирует процесс обнаружения транзиентов на астрономических изображениях. Найдите его в каталоге Chapter_5, скачанном с сайта книги. Чтобы избежать повторения кода, программа использует изображения, уже зарегистрированные кодом blink_comparator.py, поэтому вам потребуется расположить каталоги night_1_registered_transients и night_2 в директории данного проекта (см. рис. 5.3). Как и в предыдущем проекте, код Python должен находиться в директории уровнем *выше* этих каталогов.

Импорт модулей и присваивание константы

Код листинга 5.8 импортирует модули, необходимые для запуска программы, и назначает константу отступа для обработки краевых эффектов (рис. 5.11). Отступ — это небольшое расстояние, которое измеряют перпендикулярно краям

изображения и которое нужно исключить из анализа. Любые объекты, обнаруженные между краем изображения и линией отступа, игнорируются.

Листинг 5.8. Импорт модулей и определение константы, чтобы избежать краевых эффектов

transient_detector.py, часть 1

```
import os
from pathlib import Path
import cv2 as cv

PAD = 5 # Игнорировать пиксели вплоть до этого расстояния от края
```

Нам понадобятся все модули, использованные в предыдущем проекте, кроме `NumPy`, поэтому импортируем их сюда. Устанавливаем для отступа расстояние в 5 пикселей. При применении различных датасетов это значение может немного изменяться. Позже мы нарисуем прямоугольник вокруг области края внутри изображения, чтобы можно было видеть, какую область этот параметр исключает.

Обнаруживаем и обводим транзиенты

В листинге 5.9 определяем функцию, которую мы используем, чтобы найти и обвести кружком до двух транзиентов в каждой паре изображений. Транзиенты в области отступа функция будет игнорировать.

Листинг 5.9. Определение функции, которая позволит обнаружить и обвести кружком транзиенты

transient_detector.py, часть 2

```
def find_transient(image, diff_image, pad):
    """Найдем и обведем кружком транзиенты, движущиеся в звездном небе"""
    transient = False
    height, width = diff_image.shape
    cv.rectangle(image, (PAD, PAD), (width - PAD, height - PAD), 255, 1)
    minVal, maxVal, minLoc, maxLoc = cv.minMaxLoc(diff_image)
❶   if pad < maxLoc[0] < width - pad and pad < maxLoc[1] < height - pad:
        cv.circle(image, maxLoc, 10, 255, 0)
        transient = True
    return transient, maxLoc
```

Функция `find_transient()` содержит три параметра: входное изображение, изображение, показывающее отличие между первым и вторым входными изображениями (*разностную карту*), и константу `PAD`. Эта функция находит расположение самого яркого пикселя на разностной карте, рисует вокруг него кружок и возвращает его локацию вместе с логическим значением, указывающим на обнаружение объекта.

Начинаем функцию с установки переменной `transient` как `False`. Она обнаруживает транзиент. Поскольку в реальной жизни транзиенты удается найти редко, базовым состоянием этой переменной должно быть `False`.

Чтобы применить константу `PAD` и исключить область вдоль краев изображения, нам нужны его границы. Их мы получаем с помощью атрибута `shape`, который возвращает кортеж с высотой и шириной изображения.

Переменные `height` и `weight`, а также константа `PAD` используются для изображения белого прямоугольника на переменной `image` с помощью метода OpenCV `rectangle()`. Позже он будет показывать пользователю, какие части изображения были проигнорированы.

Переменная `diff_image` является массивом `NumPy`, представляющим пиксели. Фон черный, и любые «звезды», изменившие положение (или появившиеся из ниоткуда) между двумя входными изображениями, будут серыми или белыми (рис. 5.12).

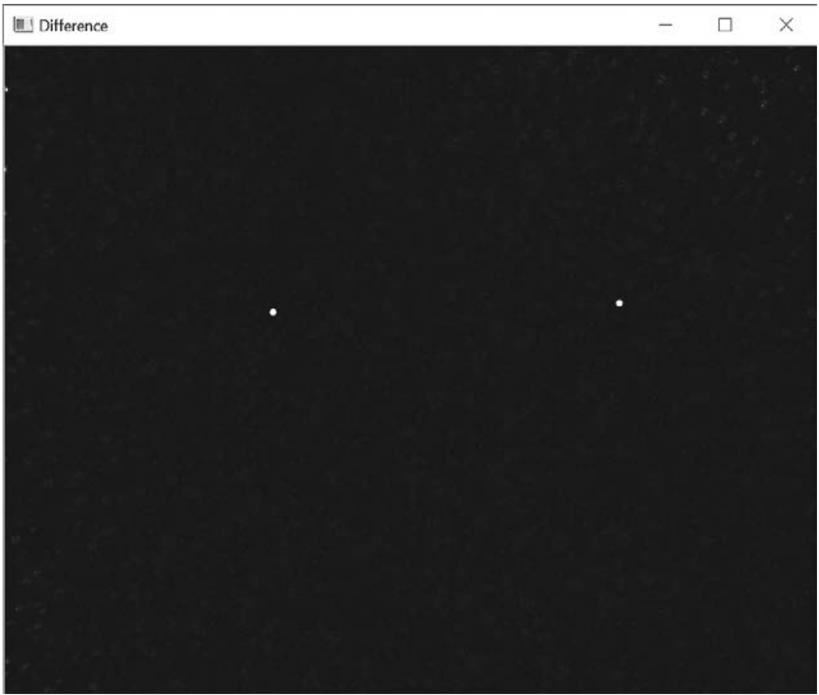


Рис. 5.12. Изображение отличий, полученное из входных изображений «яркого транзиента»

Чтобы обнаружить самый яркий наблюдаемый транзиент, используем метод OpenCV `minMaxLoc()`, который возвращает минимальное и максимальное значения пикселей на изображении вместе с кортежем их местоположения. Обратите внимание, что я называю переменные согласно схеме именования OpenCV, основанной на смешанных регистрах (например, `maxLoc`). Если вы хотите использовать другой способ, более приемлемый для руководства по стилю Python PEP8 (https://www.python.org/dev/peps/pep-0008/), то ничто не мешает вам писать, например, `max_loc` вместо `maxLoc`.

Мы можем обнаружить максимальное значение возле края изображения, поэтому с помощью условия нужно исключить подобный случай. Для этого проигнорируем значения, которые найдены в области, обозначенной константой `PAD` ❶. Если же местоположение такую проверку проходит, мы его обводим кружком на переменной `image`. Для этого используется белая окружность радиусом в 10 пикселей и толщиной линии 0.

Если вы нарисовали кружок, значит, был найден транзиент, поэтому переменная `transient` становится `True`. Это вызывает дополнительную активность программы на более поздних этапах.

Завершаем функцию возвращением переменных `transient` и `maxLoc`.

> **ПРИМЕЧАНИЕ**
>
> Метод minMaxLoc() чувствителен к шумам и грешит ложноположительными срабатываниями, поскольку работает с отдельными пикселями. Обычно сначала выполняется этап предварительной обработки, например размытие, который удаляет сомнительные пиксели. Однако в результате далее возможен пропуск астрономических объектов, которые иногда трудно отличить от шума на одном изображении.

Подготовка файлов и каталогов

Код листинга 5.10 определяет функцию `main()`, создает список имен файлов во входных каталогах и присваивает пути этих каталогов переменным.

Листинг 5.10. Определение main(), перечисление содержимого каталогов и присваивание переменных их путям

transient_detector.py, часть 3

```
def main():
    night1_files = sorted(os.listdir('night_1_registered_transients'))
    night2_files = sorted(os.listdir('night_2'))
    path1 = Path.cwd() / 'night_1_registered_transients'
    path2 = Path.cwd() / 'night_2'
    path3 = Path.cwd() / 'night_1_2_transients'
```

Определяем функцию `main()`. Далее, так же как делали в листинге 5.2 на с. 141, перечисляем содержимое каталогов, в которых хранятся входные изображения, и присваиваем их пути переменным. Для хранения изображений, содержащих обнаруженные транзиенты, мы будем использовать существующий каталог.

Перебор изображений и вычисление абсолютных разниц

Листинг 5.11 запускает цикл `for` по парам изображений. Эта функция считывает соответствующие пары изображений в виде полутоновых массивов, вычисляет различие между изображениями и показывает результат в окне. Затем она вызывает функцию `find_transient()` для полученного изображения отличий.

Листинг 5.11. Перебор изображений и поиск транзиентов

transient_detector.py, часть 4

```
    for i, _ in enumerate(night1_files[:-1]): # Убираем негативное
                                                изображение
        img1 = cv.imread(str(path1 / night1_files[i]), cv.IMREAD_GRAYSCALE)
        img2 = cv.imread(str(path2 / night2_files[i]), cv.IMREAD_GRAYSCALE)

        diff_imgs1_2 = cv.absdiff(img1, img2)
        cv.imshow('Difference', diff_imgs1_2)
        cv.waitKey(2000)

        temp = diff_imgs1_2.copy()
        transient1, transient_loc1 = find_transient(img1, temp, PAD)
        cv.circle(temp, transient_loc1, 10, 0, -1)

        transient2, _ = find_transient(img1, temp, PAD)
```

Начинаем цикл `for`, перебирающий снимки в списке `night1_files`. Программа настроена на работу с *позитивными* изображениями, поэтому возьмем срез изображения (`[:-1]`) для исключения негативных. Для получения счетчика применяем `enumerate()` и называем его не `_`, а `i`, поскольку позже он будет использоваться в качестве индекса.

Для нахождения различий между изображениями просто вызываем метод `cv.absdiff()` и передаем ему переменные для этих двух изображений. Показываем результат в течение двух секунд, после чего продолжаем выполнение программы.

Поскольку мы собираемся приглушить самый яркий транзиент, то сначала делаем копию `diff_imgs1_2`. Называем ее `temp`, подразумевая ее временный характер. Теперь вызываем написанную ранее функцию `find_transient()` и передаем ей первое входное изображение, изображение отличий и константу

PAD. Полученные результаты используем для обновления переменной transient и создания новой переменной, transient_loc1, которая записывает местоположение самого яркого пикселя на изображении отличий.

Транзиент мог попасть или не попасть на оба изображения, сделанных в две последовательные ночи. Чтобы понять, оказался ли он на снимке, скрываем только что найденное яркое пятно под черным кружком. Делаем это для изображения temp, используя в качестве цвета черный и толщину линии -1, указав таким образом OpenCV на необходимость закрасить весь кружок. Продолжаем использовать радиус 10, хотя можно его и уменьшить, если вы полагаете, что два транзиента окажутся очень близко друг к другу.

Снова вызываем функцию find_transient(), но используем для переменной местоположения нижнее подчеркивание, поскольку повторно она нам не понадобится. Маловероятно, что мы обнаружим два транзиента. Нахождения даже одного будет достаточно, чтобы открыть изображения для более детального изучения, поэтому не стоит озадачиваться поиском большего числа транзиентов.

Раскрытие транзиента и сохранение изображения

Листинг 5.12 продолжает цикл for функции main(). Здесь отображается первое входное изображение с обведенными транзиентами, выполняется публикация имен файлов изображений и сохранение изображения под новым именем. Помимо этого, в окне интерпретатора выводится журнал результатов для каждой пары изображений.

Листинг 5.12. Показ обведенных кружками транзиентов, логирование результатов и их сохранение

transient_detector.py, часть 5

```
        if transient1 or transient2:
            print('\nTRANSIENT DETECTED between {} and {}\n'
                  .format(night1_files[i], night2_files[i]))
          ❶ font = cv.FONT_HERSHEY_COMPLEX_SMALL
            cv.putText(img1, night1_files[i], (10, 25),
                       font, 1, (255, 255, 255), 1, cv.LINE_AA)
            cv.putText(img1, night2_files[i], (10, 55),
                       font, 1, (255, 255, 255), 1, cv.LINE_AA)

            blended = cv.addWeighted(img1, 1, diff_imgs1_2, 1, 0)
            cv.imshow('Surveyed', blended)
            cv.waitKey(2500)

          ❷ out_filename = '{}_DECTECTED.png'.format(night1_files[i][:-4])
            cv.imwrite(str(path3 / out_filename), blended) # Будет
                                                             перезаписано!
```

```
        else:
            print('\nNo transient detected between {} and {}\n'
                  .format(night1_files[i], night2_files[i]))

if __name__ == '__main__':
    main()
```

Начинаем с условия, которое проверяет, был ли найден транзиент. Если условие вычисляется как `True`, выводим в оболочку сообщение. Для четырех изображений, оцененных в цикле `for`, результат должен получиться таким:

```
TRANSIENT DETECTED between 1_bright_transient_left_registered.png
and 1_bright_transient_right.png

TRANSIENT DETECTED between 2_dim_transient_left_registered.png
and 2_dim_transient_right.png

TRANSIENT DETECTED between 3_diff_exposures_left_registered.png
and 3_diff_exposures_right.png

TRANSIENT DETECTED between 4_single_transient_left_registered.png
and 4_single_transient_right.png

No transient detected between 5_no_transient_left_registered.png
and 5_no_transient_right.png
```

Отрицательный результат говорит о том, что программа работает должным образом и сравнение изображений выполняется.

Далее размещаем имена изображений с положительным ответом в массиве `img1`. Начинаем с присваивания переменной шрифта для OpenCV ❶. Список доступных шрифтов можете поискать на странице https://docs.opencv.org/4.3.0/ по запросу *HersheyFonts*.

Теперь вызываем метод OpenCV `putNext()` и передаем ему первое входное изображение, имя файла изображения, позицию, переменную `font`, размер, цвет (белый), толщину и тип линии. Атрибут `LINE_AA` создает сглаженную линию. Повторяем этот код для второго изображения.

В случае обнаружения двух транзиентов можно их оба показать на одном изображении при помощи метода `addWeighted()` из OpenCV. Этот метод вычисляет взвешенную сумму двух массивов. Аргументами в данном случае являются первое изображение и вес, второе изображение и вес, а также скаляр, прибавляемый к каждой сумме. Используем первое входное изображение и изображение отличий, устанавливаем веса равными `1`, чтобы задействовать каждое изображение полностью, а скаляр устанавливаем на `0`. Результат присваиваем переменной `blended`.

Смешанное (blended) изображение показываем в окне Surveyed (Исследовано). На рис. 5.13 показан результат для «яркого» транзиента (bright).

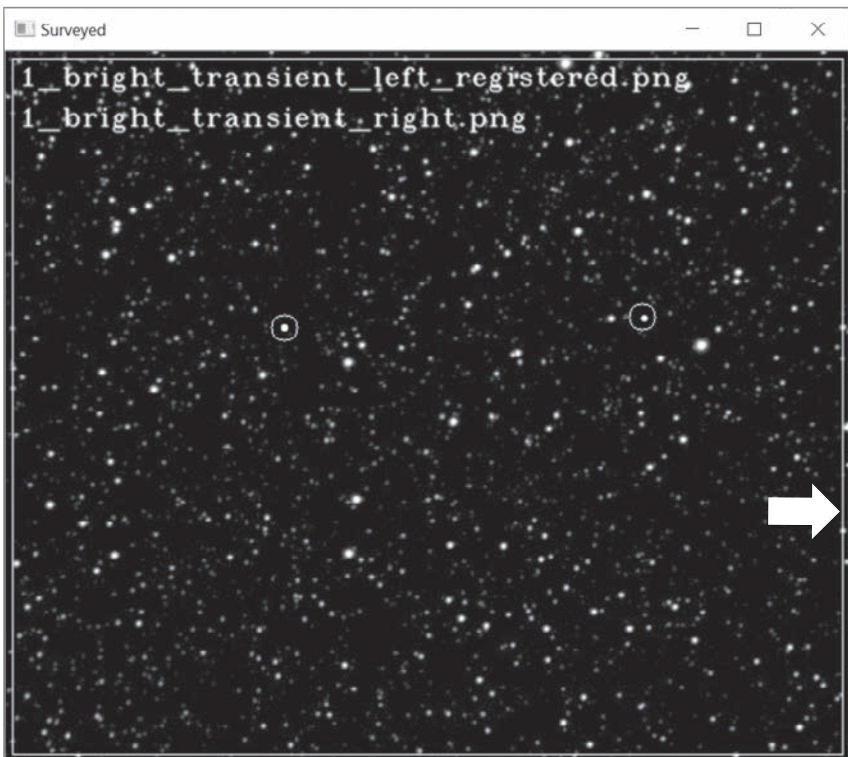


Рис. 5.13. Пример окна вывода transient_detector.py, где стрелка указывает на отступ в виде прямоугольника

Обратите внимание на белую рамку вдоль края изображения. Она показывает расстояние PAD. Все транзиенты, выходящие за этот прямоугольник, программа игнорировала.

Сохраняем это смешанное изображение, используя имя файла текущего входного изображения плюс DETECTED ❷. Тусклый транзиент на рис. 5.13 сохраняем как 1_bright_transient_left_registered_DETECTED.png. Записываем его в каталог night_1_2_transients, используя переменную `path3`.

Если транзиенты найдены не были, документируем результат в окне оболочки, после чего завершаем программу кодом для ее запуска в качестве модуля или в автономном режиме.

Использование детектора транзиентов

Вообразите, насколько обрадовался бы Клайд Томбо, будь у него вот такой детектор транзиентов. Это устройство из разряда «настрой и забудь». Даже изменение яркости на изображениях третьей пары, столь проблематичное в случае с блинк-компаратором, не представляет сложности для этой программы.

Итоги

В этой главе мы на современный лад воссоздали старую технологию, основанную на применении блинк-компаратора, после чего доработали процесс при помощи техник компьютерного зрения. Мы использовали модуль `pathlib`, чтобы упростить работу с путями каталогов, а также ввели одиночное подчеркивание для малозначимых, неиспользуемых имен переменных. Кроме того, мы применили OpenCV для поиска, описания и сопоставления интересных признаков изображений, выровняли эти признаки с помощью гомографии, смешали изображения и записали результат в файл.

Дополнительная литература

«Out of the Darkness: The Planet Pluto» (Stackpole Books, 2017), написанная самим открывателем Плутона Клайдом Томбом (Clyde Tombaugh) и его соавтором Патриком Муром (Patrick Moore), повествует, естественно, об истории обнаружения Плутона.

В книге «Chasing New Horizons: Inside the Epic First Mission to Pluto»[1] (Picador, 2018) Алан Стер (Alan Stern) и Дэвид Гринспун (David Grinspoon) рассказывают о том, как отправляли на Плутон космическое судно с прахом Клайда Томбо.

Практический проект: представление орбитальной траектории

Отредактируйте программу transient_detector.py так, чтобы в случае наличия транзиента на обоих входных снимках OpenCV рисовала линию, соединяющую эти транзиенты. Таким образом удастся обозначить орбитальную траекторию транзиента относительно звезд на заднем плане.

Именно эта информация оказалась ключом для обнаружения Плутона. Клайд Томбо использовал расстояние между двумя положениями планеты на двух

[1] Стерн А., Гринспун Д. «За новыми горизонтами. Первый полет к Плутону».

фотопластинах, а также время между экспозициями, дабы убедиться, что планета проходит рядом с прогнозированной Лоуэллом траекторией, а также чтобы удостовериться, что это именно планета, а не астероид, вращающийся вокруг Земли и поэтому расположенный на более близком расстоянии от Земли.

Решение под названием practice_orbital_path.py вы можете найти в приложении к книге или в каталоге Chapter_5.

Практический проект: найди отличия

Сопоставление признаков, которым мы занимались в этой главе, широко применяется не только в астрономии. К примеру, морские биологи используют схожие техники для идентификации особей китовых акул по их пятнам, что повышает точность подсчета популяции.

На рис. 5.14 левый и правый снимки различаются. Можете найти отличия? Давайте даже так. Сможете ли вы написать программы Python, которые будут выравнивать и сравнивать эти изображения, а также обводить кружками найденное изменение?

Рис. 5.14. Найдите отличие между левым и правым изображениями

Исходные изображения вы найдете в каталоге montages директории Chapter_5, доступной для скачивания с сайта книги. Это цветные изображения, которые, прежде чем искать объекты, нужно конвертировать в оттенки серого и выровнять.

Решения, называемые practice_montage_aligner.py и practice_montage_difference_finder.py, вы найдете в приложении к книге и в каталоге montages.

Усложняем проект: сосчитаем звезды

Как пишет журнал *Sky and Telescope*, невооруженным глазом в любом полушарии звездного неба можно видеть 9906 звезд (https://www.skyandtelescope.com/astronomy-resources/how-many-stars-night-sky-09172014/). Это безусловно много, но если взглянуть в телескоп, то их число возрастет экспоненциально.

Для оценки количества звезд астрономы наблюдают за небольшими участками неба. Они используют программу для подсчета звезд, а затем экстраполируют результаты на более обширные области. Представьте, что вы работаете ассистентом в обсерватории Лоуэлла и занимаетесь наблюдениями. Напишите программу Python, подсчитывающую количество звезд на изображении 5_no_transient_left.png, использованном в проектах 7 и 8.

Подсказки ищите онлайн по запросу *«как подсчитать точки в изображении с помощью Python и OpenCV»* (how to count dots in an image with Python and OpenCV). Решение, использующее Python и SciPy, вы найдете на странице http://prancer.physics.louisville.edu/astrowiki/index.php/Image_processing_with_Python_and_SciPy. Вы добьетесь более точных результатов, если разделите изображение на меньшие части.

6

Победа в лунной гонке с помощью «Аполлона-8»

Летом 1968 года США проигрывали лунную гонку. Советский космический аппарат «Зонд» уже был готов к отправке на Луну. ЦРУ даже сфотографировало гигантскую советскую ракету Н-1, ожидающую запуска со стартового комплекса, что несколько удручило американцев, так как их программе «Аполлон» предстояло еще три испытательных полета. Однако в августе менеджер NASA Джордж Лоу высказал отчаянную идею, предложив отправиться на Луну прямо *сейчас*. Вместо дополнительных испытаний на орбите Земли он предложил в качестве испытания облететь в декабре Луну. В этот момент космическая гонка, по существу, была закончена. Меньше чем через год Нил Армстронг совершил высадку на Луну — во имя всего человечества.

Решение отправить «Аполлон-8» на Луну далось непросто. В 1967 году трое человек погибли в капсуле «Аполлона-1», а многие беспилотные корабли либо взрывались, либо не могли выполнить миссию по другим причинам. Ставки были высоки. Разработчики предложили принцип *свободного возврата*. Полет был спроектирован так, чтобы в случае отказа двигателя служебного модуля корабль просто обогнул Луну и вернулся на Землю, подобно бумерангу (рис. 6.1).

Сейчас мы напишем программу Python, использующую чертежную доску `turtle` для имитации траектории свободного возврата «Аполлона-8». При этом мы также проработаем одну из классических задач физики — задачу трех тел.

Рис. 6.1. Эмблема «Аполлона-8», на которой номер обозначен пролегающей вокруг Луны траекторией свободного возврата

Цель миссии «Аполлон-8»

Целью миссии «Аполлон-8» был облет Луны, поэтому не было необходимости оборудовать аппарат средствами посадки на Луне. Астронавты путешествовали в командно-сервисном модуле — *CSM (command and service module)* (рис. 6.2).

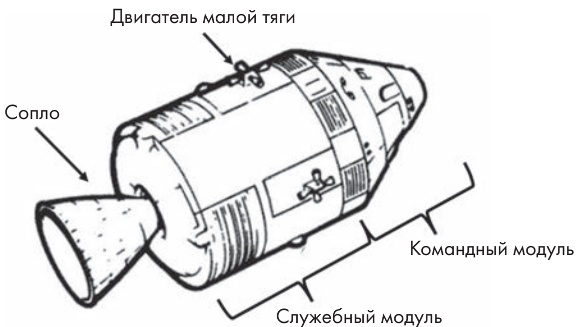


Рис. 6.2. Командно-сервисные модули «Аполлона»

К концу 1968 года двигатель CSM был протестирован только на орбите Земли, и его надежность вызывала оправданные сомнения. Для облета Луны двигатель запускался дважды: один раз — чтобы замедлить корабль для выхода на лунную орбиту, и второй раз — для того чтобы ее покинуть. Если бы первый маневр провалился, то при следовании по траектории свободного возврата астронавты

все равно вернулись бы домой. В итоге же оказалось, что двигатель отлично сработал в обоих случаях и «Аполлон-8» облетел Луну 10 раз. (Однако печально известному «Аполлону-13» пришлось-таки воспользоваться этой траекторией свободного возврата.)

Траектория свободного возврата

Вычисление траектории свободного возврата требует объемных математических вычислений. Как-никак, это ракетостроение! К счастью, можно симулировать траекторию на двухмерном графике с помощью нескольких упрощенных параметров (рис. 6.3).

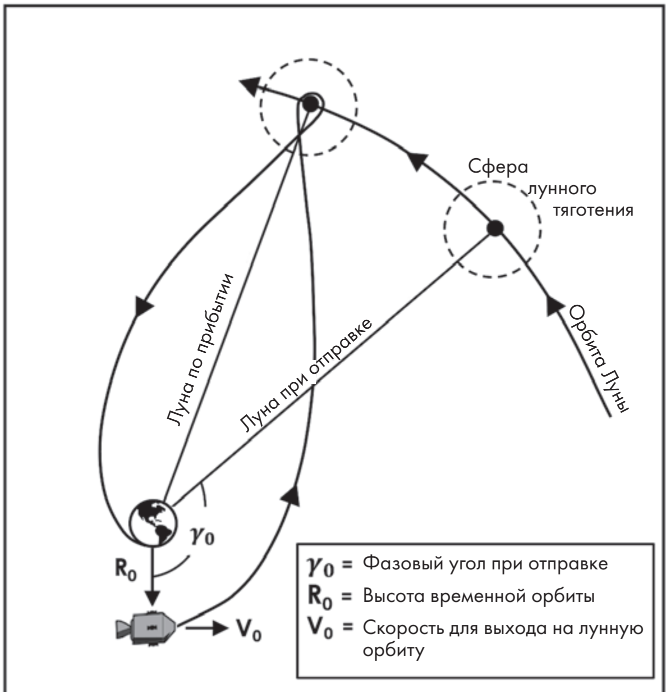


Рис. 6.3. Траектория свободного возврата (не в масштабе)

В этой двухмерной симуляции свободного возврата используется несколько ключевых значений: стартовая позиция CSM (R_0), скорость и ориентация CSM (V_0), а также фазовый угол между CSM и Луной (γ_0). *Фазовый угол*, или *угол опережения*, представляет изменение орбитальной временно́й позиции CSM, необходимое для попадания из стартовой точки в конечную. *Скорость выхода на*

транслунную орбиту (V_0) — это толкающий маневр, используемый для отправки CSM по траектории к Луне. Он осуществляется со временной околоземной орбиты, где судно выполняет внутренние проверки и ожидает оптимального фазового угла с Луной. В этот момент срабатывает и отпадает третья ступень ракеты *«Сатурн-5»*, отправляя CSM по направлению к Луне.

Поскольку Луна движется, то, прежде чем выполнять выход на транслунную орбиту, нужно спрогнозировать ее будущее положение, иначе говоря, *упредить* ее. Это можно сравнить со стрельбой по летящим тарелочкам. Для этого потребуется знать фазовый угол (γ_0) в момент выхода на транслунную орбиту. Хотя упреждение Луны несколько отличается от стрельбы по тарелкам, так как пространство в космосе искривлено и вам нужно учесть влияние гравитации Земли и Луны. Притягивающее воздействие этих двух небесных тел на космический корабль создает пертурбации, которые сложно просчитать. Дело это настолько сложное, что даже получило собственное имя в области физики — задача трех тел.

Задача трех тел

Задача трех тел — попытка спрогнозировать поведение трех взаимодействующих тел. Гравитационное уравнение Исаака Ньютона отлично работает для прогнозирования поведения двух движущихся по орбите тел, например Земли и Луны, но расчеты сильно усложняются при появлении третьего участника, будь то космический корабль, комета или нечто подобное. Ньютон так и не смог выразить простым уравнением поведение трех или более тел. На протяжении 275 лет, несмотря на предлагаемые королями награды, величайшие математики тщетно бились над этой задачей.

Проблема в том, что задачу трех тел нельзя решить, используя простые алгебраические выражения или интегралы. Вычисление влияния нескольких гравитационных полей требует численного итерирования в масштабах, которые не удается охватить без высокоскоростного компьютера наподобие вашего ноутбука.

В 1961 году Майкл Минович (Michael Minovitch), интерн в Лаборатории реактивного движения, нашел первое численное решение при помощи мейнфрейма IBM 7090, на тот момент самого быстрого компьютера в мире. Он выяснил, что можно уменьшить объем вычислений, необходимых для решения ограниченной задачи трех тел, наподобие задачи с Землей — Луной — CSM, используя модель *сопряженных конических сечений*.

Аналитическая аппроксимация *модели сопряжения конических сечений* предполагает, что вы решаете одну простую задачу для двух тел, когда космический аппарат находится в гравитационном поле Земли, и другую, отдельную задачу

для двух тел — когда этот аппарат находится в гравитационном поле Луны. Это грубое, упрощенное вычисление обеспечивает разумную оценку условий отправки и прибытия, снижая количество вариантов для начальных векторов скорости и позиции. Остается только уточнить траекторию полета с помощью повторяющихся компьютерных симуляций.

Поскольку исследователи уже нашли и задокументировали решение с сопряжением конических сечений, примененное в миссии «Аполлон-8», вычислять нам его не придется. Я адаптировал его к 2D-сценарию, который мы сейчас реализуем. Тем не менее вы вполне можете позже поэкспериментировать с альтернативными решениями, изменяя параметры R_0 и V_0, и повторно выполнить симуляцию.

Проект #9. На Луну с «Аполлоном-8»!

Представьте себя интерном в NASA, которого попросили создать простую симуляцию траектории свободного возврата «Аполлона-8», чтобы показать прессе и общественности. Поскольку NASA всегда ограничено в средствах, вам нужно использовать открытое ПО и завершить проект максимально быстро и дешево.

ЗАДАЧА

Написать программу на Python, которая графически симулирует траекторию свободного возврата, предложенную для миссии «Аполлон-8».

Использование модуля turtle

Для симуляции полета «Аполлона-8» нам нужно разработать способ рисовать и перемещать изображения на экране. Существует множество сторонних модулей, которые для этого сгодятся, но мы пойдем самым простым путем и используем предустановленный модуль `turtle`. Несмотря на то что изначально он создавался для помощи подросткам, осваивающим программирование, `turtle` удается легко адаптировать и для более серьезного применения.

Этот модуль позволяет использовать команды Python для перемещения небольшого указателя, называемого *turtle (черепаха)*, по экрану. Вы можете выбрать невидимый указатель, фактическое изображение, один из предустановленных вариантов, показанных на рис. 6.4, или настроить указатель самостоятельно.

Можно задать, чтобы при движении черепаха оставляла за собой след, позволяющий контролировать ее перемещение (рис. 6.5).

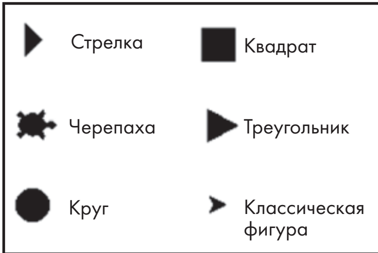


Рис. 6.4. Стандартные указатели, предлагаемые модулем turtle

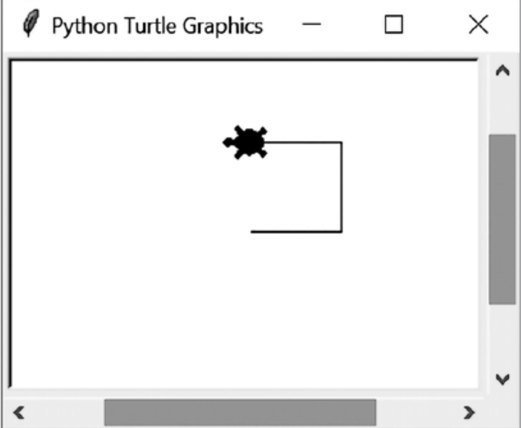


Рис. 6.5. Перемещение черепахи в окне Turtle Graphics

Это простое рисование было реализовано с помощью следующего скрипта:

```
>>> import turtle
>>> steve = turtle.Turtle('turtle') # Создает объект turtle в форме черепахи
>>> steve.fd(50) # Перемещает черепаху на 50 пикселей вперед
>>> steve.left(90) # Поворачивает черепаху влево на 90 градусов
>>> steve.fd(50)
>>> steve.left(90)
>>> steve.fd(50)
```

С помощью `turtle` можно использовать функциональность Python, что позволит писать более сжатый код. К примеру, для создания такого же шаблона можно задействовать цикл `for`.

```
>>> for i in range(3):
        steve.fd(50)
        steve.left(90)
```

Здесь `steve` перемещается на 50 пикселей вперед, а затем поворачивает влево под нужным углом. Эти шаги повторяются трижды циклом `for`.

Другие опции позволяют изменять фигуру черепахи, ее цвет, отключать рисование пути, «отпечатывать» ее текущее положение на экране, устанавливать направление движения черепахи и получать координаты ее положения. На рис. 6.6 иллюстрируется эта функциональность, прописанная в следующем за рисунком скрипте.

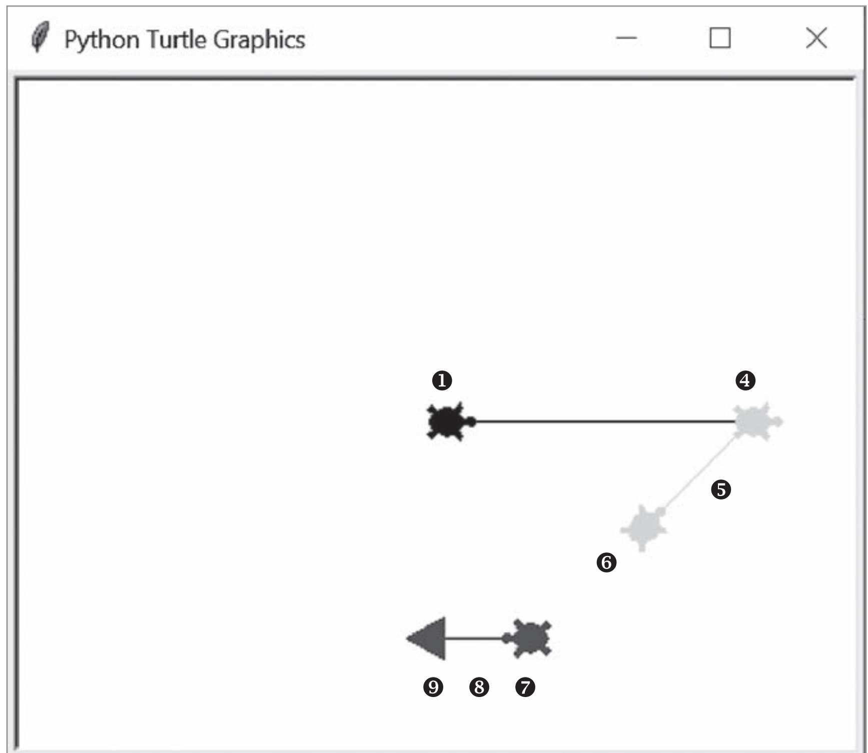


Рис. 6.6. Опции настройки указателя-черепахи. Числа указывают аннотации скрипта

```
   >>> import turtle
   >>> steve = turtle.Turtle('turtle')
❶  >>> a_stamp = steve.stamp()
❷  >>> steve.position()
    ❸ (0.00,0.00)
   >>> steve.fd(150)
❹  >>> steve.color('gray')
   >>> a_stamp = steve.stamp()
   >>> steve.left(45)
```

```
❺ >>> steve.bk(75)
  >>> a_stamp = steve.stamp()
❻ >>> steve.penup()
  >>> steve.bk(75)
  >>> steve.color('black')
❼ >>> steve.setheading(180)
  >>> a_stamp = steve.stamp()
❽ >>> steve.pendown()
  >>> steve.fd(50)
❾ >>> steve.shape('triangle')
```

После импорта модуля `turtle` и инстанцирования объекта `steve` оставляем отпечаток изображения `steve` при помощи метода `stamp()` ❶.

Далее используем метод `position()` ❷ для получения текущих координат черепахи (x, y) в качестве кортежа ❸. Они пригодятся нам при вычислении расстояния между объектами для гравитационного уравнения.

Перемещаем черепаху вперед на 150 пробелов и изменяем ее цвет на серый ❹. Затем оставляем отпечаток, поворачиваем черепаху на 45 градусов и задним ходом перемещаем на 75 пробелов, используя метод `bk()` ❺.

Оставляем еще один отпечаток, после чего с помощью метода `penup()` ❻ отключаем рисование пройденного черепахой пути. Перемещаем `steve` задним ходом еще на 75 пробелов и закрашиваем его в черный. Теперь используем альтернативу `rotate()`, которая подразумевает движение черепахи прямо ❼. Обратите внимание, что предустановленные в «стандартном режиме» направления движения отсчитываются от востока, а не от севера (табл. 6.1).

Таблица 6.1. Направления движения модуля turtle в стандартном режиме

Градусы	**Направление**
0	Восток
90	Север
180	Запад
270	Юг

Оставляем еще один отпечаток и вновь активируем отображение следа ❽. Перемещаем `steve` на 50 пробелов вперед и изменяем фигуру указателя с черепахи на треугольник ❾. На этом рисование завершается.

Выглядит довольно просто, но это лишь пример. Используя правильные команды, можно рисовать весьма замысловатые узоры, такие как мозаика Пенроуза, показанная на рис. 6.7.

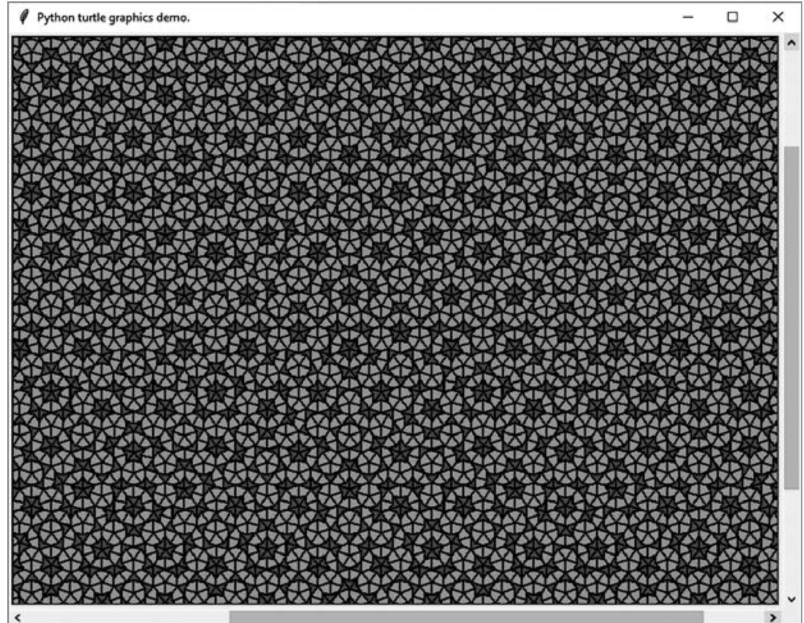


Рис. 6.7. Мозаика Пенроуза, созданная демонстрацией модуля turtle, penrose.py

Модуль `turtle` является частью Python Standard Library, и его официальную документацию вы найдете на странице https://docs.python.org/3/library/turtle.html?highlight=turtle#module-turtle/. В качестве краткого руководства поищите онлайн «Simple Turtle for Python», написанное Элом Свейгартом.

Стратегия

Итак, мы приняли стратегическое решение рисовать симуляцию с помощью `turtle`, но как же должна выглядеть эта симуляция? Для удобства я предлагаю взять за основу рис. 6.3. Начнем мы с CSM, расположенного на той же временной орбите вокруг Земли (R_0), и Луной, двигающейся с тем же аппроксимированным фазовым углом (γ_0). Луну и Землю можно представить с помощью изображений, а для создания CSM используем собственные фигуры `turtle`.

Еще одно важное решение — выбор процедурного либо объектно-ориентированного программирования (ООП). Когда планируется генерация нескольких объектов, которые ведут себя схожим образом и взаимодействуют друг с другом, лучше применять ООП. В этом случае можно использовать класс в качестве шаблона для объектов Земли, Луны, а также CSM и автоматически обновлять атрибуты объекта по ходу выполнения симуляции.

Для симуляции задействуем *временные шаги*. По сути, каждый цикл программы будет представлять одну единицу безразмерного времени. С каждым циклом нам понадобится вычислять позиции всех объектов и обновлять (перерисовывать) объекты на экране. Для этого придется решить задачу трех тел. К счастью, ее не только уже решили до нас, но и сделали это с помощью `turtle`.

Модули Python зачастую включают скрипты, демонстрирующие использование продукта. К примеру, галерея `matplotlib` хранит фрагменты кода и руководства для составления огромного числа графиков и чертежей. Также в модуле `turtle` есть turtle-example-suit, который содержит демонстрации применения `turtle`.

Одна из демонстраций, planet_and_moon.py, предоставляет интересный «рецепт» для решения задачи трех тел в `turtle` (рис. 6.8). Чтобы увидеть демо, откройте PowerShell либо терминал и введите **`python -m turtledemo`**. В зависимости от платформы и установленной версии Python вам может потребоваться использовать **`python3 -m turtledemo`**.

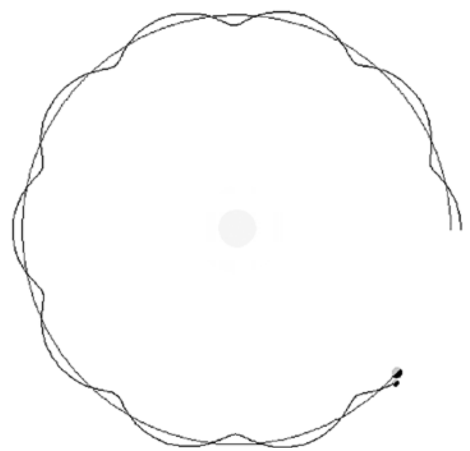


Рис. 6.8. Демонстрация turtle planet_and_moon.py

Демо решает задачу трех тел для Солнца — Земли — Луны, но его можно легко адаптировать под задачу Земля — Луна — CSM. Опять же для конкретного случая с «Аполлоном-8» в качестве ориентира при разработке программы мы используем рис. 6.3.

Код программы для расчета свободного возврата «Аполлона-8»

Программа Apollo_8_free_return.py с помощью графики `turtle` рисует в проекции вид сверху процесс отправки «Аполлона-8» с земной орбиты, его круговой полет

вокруг Луны и возвращение на Землю. Основана программа на демонстрации planet_and_moon.py, упомянутой в предыдущем разделе.

Найти программу можно в каталоге Chapter_6, доступном для скачивания с сайта книги по адресу https://nostarch.com/real-world-python/. Кроме этого, вам понадобятся изображения Земли и Луны, расположенные там же (рис. 6.9). Обязательно разместите их в одном каталоге с кодом, не меняя имен.

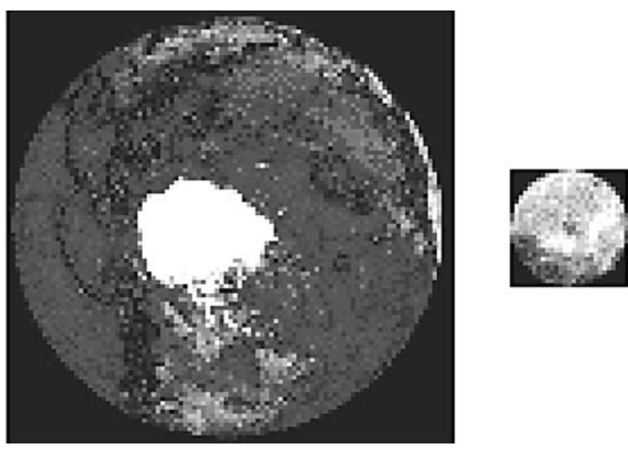


Рис. 6.9. Изображения earth_100x100.gif и moon_27x27.gif, используемые в симуляции

Импорт turtle и присваивание констант

Код листинга 6.1 импортирует модуль `turtle` и присваивает константы, представляющие ключевые параметры: гравитационную постоянную, количество выполнений основного цикла, а также значения x и y для R_0 и V_0 (см. рис. 6.3). Перечисление всех этих значений ближе к началу программы упрощает их поиск и дальнейшее изменение.

Листинг 6.1. Импорт turtle и назначение констант

apollo_8_free_return.py, часть 1

```
from turtle import Shape, Screen, Turtle, Vec2D as Vec

# Пользовательский ввод:
G = 8
NUM_LOOPS = 4100
Ro_X = 0
Ro_Y = -85
Vo_X = 485
Vo_Y = 0
```

Из `turtle` нужно импортировать четыре вспомогательных класса. Класс `Shape` мы используем для создания собственного указателя, который будет выглядеть как CSM. Подкласс `Screen` создает экран, называемый *рисовальной доской* (drawing board) на манер `turtle`. Подкласс `turtle` создает объекты указателей. Импорт `Vec2D` является классом двухмерных векторов, который поможет определять скорость в виде вектора, характеризуемого величиной и направлением.

Далее присваиваем переменные, которые пользователь сможет при желании настроить. Начинаем с гравитационной постоянной, используемой в гравитационном уравнении Ньютона для обеспечения правильности единиц измерения. Присваиваем ей 8, значение, которое применяется в демонстрации `turtle`. Рассматривайте ее как *масштабированную* гравитационную постоянную. Использовать истинную постоянную нельзя, поскольку в симуляции не применяются реальные единицы измерения.

Симуляцию мы будем выполнять циклически, где каждая итерация — шаг во времени. С каждым шагом программа будет пересчитывать позицию CSM по мере его движения через гравитационные поля Земли и Луны. Значение `4100`, полученное методом проб и ошибок, остановит симуляцию сразу после прибытия космического корабля обратно на Землю.

В 1968 году путешествие вокруг Луны занимало около шести дней. Поскольку мы с каждым циклом инкрементируем единицу времени на 0.001 и выполняем 4100 циклов, это означает, что временной шаг в симуляции представляет около двух минут реального времени. Чем длиннее временной шаг, тем быстрее симуляция, но менее точны ее результаты, так как небольшие ошибки суммируются. При реальном моделировании полета можно оптимизировать размер временного шага, сначала сделав его небольшим для максимальной точности, а затем, ориентируясь на полученный результат, найти самый большой шаг, дающий такой же результат.

Следующие две переменные, `Ro_X` и `Ro_Y`, представляют координаты CSM (x, y) во время выхода на транслунную орбиту (рис. 6.3). Аналогично этому, `Vo_X` и `Vo_Y` представляют x- и y-составляющие направления скорости выхода на эту орбиту, которые применяет третья ступень ракеты *«Сатурн-5»*. Эти значения изначально были выбраны наугад и после доработаны на повторяющихся симуляциях.

Создание гравитационной системы

Так как Земля, Луна и CSM формируют непрерывно взаимодействующую гравитационную систему, нам нужен удобный способ представить их и соответствующие им силы. Нам понадобятся два класса: один — для создания гравитационной системы и второй — для создания тел внутри нее. Код листинга 6.2

определяет класс GravSys, который поможет создать миниатюрную Солнечную систему. Этот класс с помощью списка отслеживает все движущиеся тела и перебирает их в течение серии временных шагов. Основан GravSys на демонстрации planet_and_moon.py из библиотеки turtle.

Листинг 6.2. Определение класса для управления телами в гравитационной системе

apollo_8_free_return.py, часть 2

```
class GravSys():
    """Выполняем гравитационную симуляцию для n тел."""

    def __init__(self):
        self.bodies = []
        self.t = 0
        self.dt = 0.001
❶   def sim_loop(self):
        """Прогоняем тела из списка через временные шаги."""
        for _ in range(NUM_LOOPS):
            self.t += self.dt
            for body in self.bodies:
                body.step()
```

Класс GravSys определяет продолжительность симуляции, время между временными шагами (циклами) и задействованные тела. Он также вызывает метод step() класса Body, который мы определим в листинге 6.3. Этот метод обновляет позицию каждого тела, возникающую вследствие гравитационного ускорения.

Определяем метод инициализации и, следуя соглашению, передаем ему в качестве параметра self. Параметр self представляет объект GravSys, который мы пропишем позже в функции main().

Создаем пустой список под названием bodies, где будут содержаться объекты для Земли, Луны и CSM. Далее присваиваем атрибуты, определяющие начало симуляции и количество времени, прибавляемое с каждым циклом, — *прирост времени*, или dt. Стартовое время устанавливаем как 0, а dt задаем как 0.001. Я уже говорил, что этот временной шаг будет соответствовать примерно двум минутам реального времени и производить ровную, точную и быструю симуляцию.

Последний метод управляет временными шагами симуляции ❶. Он задействует цикл for, диапазон которого установлен как переменная NUM_LOOPS. Вместо i используем одиночное подчеркивание (_), указывая на использование незначительной переменной (подробнее в листинге 5.3 главы 5).

С каждым циклом инкрементируем переменную времени гравитационной системы на величину dt. Затем применяем этот временной сдвиг к каждому телу, перебирая их список и вызывая метод body_step(), который определим позже

внутри класса Body. Этот метод обновляет позиции и скорости тел согласно гравитационному притяжению.

Создание небесных тел

Код листинга 6.3 определяет класс, на основе которого мы создадим объекты Body для Земли, Луны и CSM. Несмотря на то что никто никогда не спутал бы планету с небольшим космическим кораблем, с гравитационной точки зрения они не столь различны, и мы можем строить их по одному образцу.

Листинг 6.3. Определение класса для создания объектов для Земли, Луны и CSM

apollo_8_free_return.py, часть 3

```
class Body(Turtle):
    """Небесный объект, вращающийся по орбите и проецирующий гравитационное
    поле"""
    def __init__(self, mass, start_loc, vel, gravsys, shape):
        super().__init__(shape=shape)
        self.gravsys = gravsys
        self.penup()
        self.mass = mass
        self.setpos(start_loc)
        self.vel = vel
        gravsys.bodies.append(self)
        #self.resizemode("user")
        #self.pendown() # Раскомментируйте, чтобы рисовать путь позади
         объекта
```

Определяем новый класс, используя имеющийся класс Turtle в качестве *предка*. Это означает, что класс Body будет наследовать все методы и атрибуты Turtle.

Далее определяем метод инициализации для объекта тела. Его мы будем использовать для создания в симуляции новых объектов Body. В ООП этот процесс называется *инстанцированием*. В качестве параметров метод инициализации получает сам себя, атрибут массы, начальную позицию, начальную скорость, объект гравитационной системы и фигуру.

Функция super() позволяет вызывать метод суперкласса для получения доступа к методам, наследованным из класса-предка. Это позволяет объектам Body использовать атрибуты из ранее созданного класса Turtle. Передаем ему атрибут shape, который позволит задавать собственную фигуру или изображение тел при их построении в функции main().

Теперь присваиваем объекту Body атрибут экземпляра, который позволит гравитационной системе и телу взаимодействовать. Обратите внимание, что лучше инициализировать атрибуты через метод _init_(), как мы и делаем

в данном случае, поскольку это первый метод, вызываемый после создания объекта. Таким образом, эти атрибуты будут сразу же доступны любому методу данного класса, а другие разработчики смогут увидеть список всех атрибутов в одном месте.

Далее метод `penup()` класса `Turtle` будет отключать рисование, чтобы объект не оставлял за собой след при перемещении. Это дает возможность выполнять симуляцию с видимыми или невидимыми траекториями орбит.

Инициализируем для тела атрибут `mass`. Он потребуется для вычисления силы притяжения. Далее с помощью метода `setpos()` класса `Turtle` указываем начальную позицию тела. Начальная позиция каждого тела представлена кортежем (x, y). Исходная точка $(0, 0)$ находится в центре экрана. Значение координаты x увеличивается вправо, а y — вверх.

Присваиваем атрибут инициализации для скорости. Он будет хранить начальную скорость каждого объекта. Скорость CSM будет в течение симуляции изменяться ввиду прохождения корабля через гравитационные поля Земли и Луны.

По мере создания каждого тела используйте точечную нотацию для его добавления в список тел гравитационной системы. Объект `gravsys` мы создадим в функции `main()` из класса `GravSys()`.

Две последние, пока закомментированные, строки позволяют пользователю изменять размер окна симуляции и включать рисование пути для каждого объекта. Начинаем мы в полноэкранном режиме при отключенном рисовании пути, чтобы ускорить выполнение симуляции.

Вычисление гравитационного ускорения

Симуляция полета «Аполлона-8» начнется сразу же после выхода на транслунную орбиту. К этому моменту третья ступень *«Сатурна-5»* только что сработала и отпала, а CSM начинает свой полет к Луне. Все изменения скорости или направления будут полностью обусловлены изменениями гравитации.

Метод из листинга 6.4 перебирает тела в списке, вычисляет гравитационное ускорение для каждого из них и возвращает вектор, представляющий ускорение тела в направлениях x и y.

Листинг 6.4. Вычисление гравитационного ускорения

apollo_8_free_return.py, часть 4

```
    def acc(self):
        """Вычисляем совместное действие сил на тело и возвращаем компоненты
        вектора"""
        a = Vec(0, 0)
```

```
        for body in self.gravsys.bodies:
            if body != self:
                r = body.pos() - self.pos()
                a += (G * body.mass / abs(r)**3) * r
        return a
```

Продолжаем класс `Body`. Теперь мы определяем метод ускорения, называемый `acc()`, и передаем ему `self`. Внутри этого метода создаем локальную переменную `a`, опять же для ускорения, и присваиваем ее кортежу векторов, используя вспомогательный класс `Vec2D`. 2D-вектор — это пара вещественных чисел (a, b), которые в этом случае представляют компоненты x и y соответственно. Вспомогательный класс `Vec2D` задает правила, которые позволяют использовать арифметические операции с использованием векторов:

- $(a, b) + (c, d) = (a + c, b + d)$
- $(a, b) - (c, d) = (a - c, b - d)$
- $(a, b) \times (c, d) = ac + bd$

Далее начинаем перебирать в списке `bodies` элементы Земля, Луна и CSM. Используя силу гравитации каждого тела, мы определим ускорение объекта, для которого вызываем метод `acc()`. Ускорение телом самого себя бессмысленно, поэтому мы исключаем тело, если оно совпадает с `self`.

Для вычисления гравитационного ускорения (хранящегося в переменной `g`) в некой точке космического пространства используется следующая формула:

$$g = \frac{GM}{r^2}\hat{r}.$$

Здесь M — масса притягивающего тела, r — расстояние (радиус) между телами, G — определенная ранее гравитационная постоянная, а $\hat{r}$ — это единичный вектор из центра масс притягивающего тела к центру масс ускоряемого тела. *Единичный вектор* также называется *вектором направления* или *нормализованным вектором* и может быть описан как $r/|r|$ или:

$$\frac{(\text{Положение притягивающего тела} - \text{положение притягиваемого тела})}{|(\text{Положение притягивающего тела} - \text{положение притягиваемого тела})|}.$$

Единичный вектор фиксирует направление ускорения, которое будет либо положительным, либо отрицательным. Для вычисления этого вектора нужно определить расстояние между телами, используя метод `turtlepos()`, который позволяет получить текущую позицию каждого тела в виде вектора `Vec2D`. Как я уже говорил, это кортеж координат (x, y).

Далее мы вводим данный кортеж в уравнение ускорения. При каждом переборе нового тела переменная `a` изменяется в соответствии с силой притяжения рассматриваемого тела. К примеру, в некий момент гравитация Земли может замедлять движение CSM, а гравитация Луны может притягивать его к себе и тем самым ускорять. Переменная `a` в конце каждого цикла будет фиксировать их суммарный эффект. Завершается метод возвращением `a`.

Проход по симуляции

Листинг 6.5 продолжает класс `Body`. Здесь определяется метод для решения задачи трех тел. Он с каждым шагом по времени обновляет позицию, ориентацию и скорость тел в гравитационной системе. Чем короче шаги, тем точнее решение, хотя и ценой вычислительной эффективности.

Листинг 6.5. Применение шага по времени и поворот CSM

apollo_8_free_return.py, часть 5

```
    def step(self):
        """Вычисление положения, ориентации и скорости тела"""
        dt = self.gravsys.dt
        a = self.acc()
        self.vel = self.vel + dt * a
        self.setpos(self.pos() + dt * self.vel)
      ❶ if self.gravsys.bodies.index(self) == 2: # Индекс 2 = CSM.
            rotate_factor = 0.0006
            self.setheading((self.heading() - rotate_factor * self.xcor()))
          ❷ if self.xcor() < -20:
                self.shape('arrow')
                self.shapesize(0.5)
                self.setheading(105)
```

Определяем метод `step()` для вычисления позиции, ориентации и скорости тела. Присваиваем ему в качестве аргумента `self`.

Внутри метода устанавливаем локальную переменную `dt` как объект `gravsys` с таким же именем. Данная переменная не имеет связи ни с какой системой реального времени. Это просто число с плавающей точкой, которое мы используем для инкрементирования скорости с каждым временным шагом. Чем больше значение переменной `dt`, тем быстрее выполняется симуляция.

Теперь вызываем метод `self.acc()` для вычисления ускорения, которое движущееся тело испытывает под действием смешанных полей гравитации других тел. Этот метод возвращает кортеж векторов с координатами (x, y). Умножаем его на `dt` и прибавляем результат к `self.vel()`, который также является вектором, обновляя тем самым скорость тела для текущего временного шага. Напомню, что векторной арифметикой руководит класс `Vec2D`.

Чтобы обновить позицию тела в графическом окне `turtle`, умножаем скорость тела на временной шаг и прибавляем результат к атрибуту позиции тела. Теперь тело двигается согласно гравитационному притяжению других тел. Мы только что решили задачу трех тел!

Далее добавляем код для уточнения поведения CSM. Тяговый двигатель расположен в задней части CSM, поэтому в реальных условиях именно кормой корабль направлен на цель. Таким образом, двигатель может запуститься и замедлить корабль в достаточной степени, чтобы тот вошел в земную атмосферу или вышел на лунную орбиту. Ориентировать корабль, летящий по траектории свободного возврата, именно так необязательно, но поскольку «Аполлон-8» планировал запустить двигатели и выйти на лунную орбиту (что он и сделал), в процессе полета нам нужно поворачивать корабль соответствующим образом.

Начинаем с выбора CSM из списка тел ❶, где он будет третьим с индексом 2.

Чтобы CSM при полете в космическом пространстве поворачивался, присваиваем небольшое число локальной переменной `rotate_factor`. Я подобрал это значение путем проб и ошибок. Далее устанавливаем направление указателя-черепахи CSM при помощи его атрибута `selfheading`. Мы не передаем ему координаты (x, y), а вызываем метод `self.heading()`, который возвращает текущее направление объекта в градусах, и вычитаем из него переменную `rotate_factor`, которую умножаем на текущее положение x тела, полученное из вызова метода `self.xcor()`. В результате по мере приближения к Луне CSM будет поворачиваться быстрее, чтобы удерживать корму в направлении цели полета.

Нам потребуется отстыковать служебный модуль до того, как корабль войдет в земную атмосферу. Чтобы сделать это в той точке, в которой это происходило в реальности при полете «Аполлона», используем еще одно условие для проверки координаты x корабля ❷. Симуляция предполагает, что Земля находится рядом с центром экрана в координатах $(0, 0)$. В `turtle` координата x уменьшается по мере движения влево от центра и увеличивается при движении вправо. Если координата x CSM меньше -20 пикселей, то можно предположить, что корабль возвращается домой и со служебным модулем пора прощаться.

Мы смоделируем это событие, изменив форму указателя `turtle`, представляющего CSM. Поскольку в `turtle` есть стандартная фигура под названием `arrow` (стрелка), которая по виду напоминает командный модуль, нужно лишь вызвать метод `self.shape()` и передать ему имя этой фигуры. Затем вызываем метод `self.shapesize()` и уменьшаем вдвое размер стрелки, чтобы она соответствовала размеру командного модуля в пользовательской фигуре CSM, которую мы создадим позднее. Когда CSM пройдет позицию x со значением -20, служебный модуль волшебным образом исчезнет, а командный модуль продолжит свой путь домой.

Теперь направим основание командного модуля с жаростойким покрытием в направлении Земли. Для этого устанавливаем направление указателя-стрелки на 105 градусов.

Определение main(), настройка экрана и инстанцирование гравитационной системы

Мы использовали объектно-ориентированное программирование для создания гравитационной системы и тел внутри нее. Для выполнения симуляции мы вернемся к процедурному программированию и используем функцию `main()`. Она настраивает графический экран `turtle`, инстанцирует объекты для гравитационной системы и трех тел, создает пользовательскую фигуру для CSM и вызывает метод `sim_loop()` гравитационной системы для выполнения всех временных шагов.

В листинге 6.6 мы определяем функцию `main()` и настраиваем экран. Здесь же мы создаем объект гравитационной системы для управления нашей мини-солнечной системой.

Листинг 6.6. Настройка экрана и создание в main() объекта gravsys

apollo_8_free_return.py, часть 6

```
def main():
    screen = Screen()
    screen.setup(width=1.0, height=1.0) # Для полноэкранного режима
    screen.bgcolor('black')
    screen.title("Apollo 8 Free Return Simulation")

    gravsys = GravSys()
```

Определяем `main()` и затем инстанцируем объект `screen` (окно для рисования) на основе подкласса `TurtleScreen`. После вызываем метод `setup()` объекта `screen`, устанавливая размер `screen` как полный экран. Для этого передаем аргументы `width` и `height` со значением `1`.

Если вы не хотите, чтобы окно для рисования занимало весь экран, передайте в `setup()` аргументы со значениями пикселей из следующего сниппета:

```
screen.setup(width=800, height=900, startx=100, starty=0)
```

Обратите внимание, что отрицательное значение `startx` выравнивается по правому краю, а отрицательное значение `starty` — по нижнему. При этом параметры по умолчанию создают центрированное окно. Можете поэкспериментировать с этими параметрами, чтобы добиться наилучшего соответствия вашему монитору.

Завершаем настройку экрана — устанавливаем его фоновый цвет как черный и прописываем название. Далее с помощью класса `GravSys` инстанцируем

объект гравитационной системы `gravsys`. Этот объект дает доступ к атрибутам и методам класса `GravSys`. Мы будем передавать его каждому телу при инстанцировании.

Создание Земли и Луны

Листинг 6.7 продолжает функцию `main()`. В нем с помощью ранее определенного класса `Body` мы создаем указатели для Земли и Луны. Земля останется в стационарной позиции в середине экрана, а Луна будет вокруг нее вращаться.

При создании этих объектов мы установим их начальные координаты. Начальную позицию Земли определим почти в центре экрана, слегка сместив, чтобы Земле и CSM было достаточно места для взаимодействия в верхней части окна.

Начальная позиция Луны и CSM должна отражать положение объектов на рис. 6.3, где CSM находится точно под центром Земли. Таким образом, нам потребуется активировать тягу только в направлении x, мы будем избавлены от вычисления компонентов вектора скорости — частично в направлении x и частично в направлении y.

Листинг 6.7. Инстанцирование указателей для Земли и Луны

apollo_8_free_return.py, часть 7

```
    image_earth = 'earth_100x100.gif'
    screen.register_shape(image_earth)
    earth = Body(1000000, (0, -25), Vec(0, -2.5), gravsys, image_earth)
    earth.pencolor('white')
    earth.getscreen().tracer(n=0, delay=0)

❶ image_moon = 'moon_27x27.gif'
    screen.register_shape(image_moon)
    moon = Body(32000, (344, 42), Vec(-27, 147), gravsys, image_moon)
    moon.pencolor('gray')
```

Начинаем с присваивания переменной изображения Земли, которое хранится в каталоге проекта. Заметьте, что изображения должны быть файлами gif и поворачивать их, чтобы задать направление указателя, нельзя. Поэтому, чтобы `turtle` распознала новую фигуру, добавьте ее в `shapelist` подкласса `TurtleScreen` при помощи метода `screen.register_shape()`. Ему следует передать переменную, ссылающуюся на изображение Земли.

Теперь пора инстанцировать объект указателя для Земли. Вызываем класс `Body` и передаем ему аргументы массы, начальной позиции, начальной скорости, гравитационной системы и фигуры указателя — в данном случае изображения Земли. Рассмотрим каждый из этих аргументов подробнее.

Здесь мы не используем реальные единицы измерения, поэтому масса указана как произвольное число. Я начал со значения, используемого для Солнца в демонстрации planet_and_moon.py, на которой вся наша программа и основана.

Начальная позиция является кортежем (x, y), который помещает изображение Земли рядом с центром экрана. При этом она оказывается смещенной вниз на 25 пикселей, так как основное действие будет происходить в верхнем квадранте экрана. Такое расположение предоставит телам больше простора именно в этой области.

Начальная скорость представлена просто кортежем (x, y), передаваемым вспомогательному классу `Vec2D` в виде аргумента. Как я уже говорил, это позволит следующим методам изменять атрибут скорости, используя векторную арифметику. Обратите внимание, что скорость Земли равна не `(0, 0)`, а `(0, -2.5)`. Как в реальности, так и в нашей симуляции Луна оказывает на Землю гравитационное воздействие, достаточное для того, чтобы сместить их общий центр тяжести в сторону от центра Земли. В результате указатель Земли в процессе симуляции будет колебаться, меняя тем самым позиции других тел. Поскольку Луна расположена в верхней части экрана, небольшое смещение Земли вниз при каждом временном шаге нивелирует эти колебания.

Последние два аргумента представляют объект `gravsys`, инстанцированный в предыдущем листинге, и переменную изображения для Земли. Передача `gravsys` означает, что указатель Земли будет добавлен в список тел и включен в метод класса `sim_loop()`.

Обратите внимание, если вы не хотите использовать много аргументов при инстанцировании объекта, то можете изменить его атрибуты после создания. Например, при определении класса `Body` вместо использования аргумента для массы можно установить `self.mass = 0`. Далее после инстанцирования Земли можно сбросить значение массы, используя `earth.mass = 1000000`.

Поскольку Земля совершает небольшие колебания, траектория ее орбиты сформирует в верхней части планеты сплошной круг. Чтобы скрыть его в полярной шапке, мы используем `turtle`-метод `pencolor()` и устанавливаем цвет линии как `white`.

В завершение откладываем запуск симуляции и исключаем мигание разных указателей в процессе, когда программа их только создает и изменяет до нужного размера. Метод `getscreen()` возвращает объект `TurtleScreen`, на котором черепаха рисует. Затем для этого объекта можно вызывать методы `TurtleScreen`. В той же строке вызываем метод `tracer()`, который включает и отключает анимацию указателя, а также устанавливает задержку для обновления в ходе рисования. Параметр n определяет количество обновлений экрана. Значение 0 означает,

что он обновляется с каждым циклом. Увеличение этого значения постепенно сокращает частоту обновлений. Это можно использовать для ускорения рисования сложной графики, но в ущерб качеству изображения. Второй аргумент устанавливает значение задержки между обновлениями экрана в миллисекундах. Увеличение задержки замедляет анимацию.

Указатель Луны мы создадим аналогично. Начнем с присваивания новой переменной для хранения изображения Луны ❶. Масса Луны составляет всего лишь несколько процентов от массы Земли, поэтому используем в ее случае намного меньшее значение. Я начал с величины около 16 000 и корректировал его, пока траектория полета CSM вокруг Луны не оказалась визуально удовлетворительной.

Начальная позиция Луны управляется фазовым углом, показанным на рис. 6.3. Как и сам рисунок, создаваемая симуляция — не в масштабе. Несмотря на то что относительные размеры изображений Земли и Луны верны, расстояние между ними не отражает истинную пропорцию, поэтому фазовый угол следует соответствующим образом подстроить. Я уменьшил это расстояние в модели, потому что космические расстояния огромны. Очень! Если вы хотите показать симуляцию в масштабе и при этом вписать ее в формат монитора, то придется использовать до невозможности мелкие тела Земли и Луны (рис. 6.10).

Рис. 6.10. Система Земли и Луны в их максимальном приближении, или перигее, показана в точном масштабе

Чтобы оба тела оставались узнаваемыми, мы используем более крупные, правильно масштабированные друг относительно друга изображения, но уменьшим расстояние между ними (рис. 6.11). Такая конфигурация удобна для зрителя и позволит показать траекторию свободного возвращения.

Поскольку Земля и Луна в этой симуляции оказываются ближе друг к другу, орбитальная скорость Луны, согласно второму закону Кеплера о движении планет, будет быстрее, чем в реальности. Чтобы это компенсировать, ее начальную позицию мы выбираем так, чтобы уменьшить фазовый угол по сравнению с показанным на рис. 6.3.

Рис. 6.11. Система Земли и Луны в симуляции, где пропорционально масштабированы только размеры тел

В завершение нам нужна опция рисования орбиты Луны. Для этого используем метод `pencolor()` и установим цвет линии на `gray`.

ПРИМЕЧАНИЕ

Такие параметры, как масса, начальная позиция и начальная скорость, хорошо подходят на роль глобальных констант. Несмотря на это, я предпочел вводить их в виде аргументов метода, чтобы не перегружать пользователя избытком входных переменных в начале программы.

Создание собственной фигуры для CSM

Пришло время инстанцировать объект указателя, представляющий CSM. Для этого потребуется немного больше усилий, чем для предыдущих двух объектов.

Во-первых, нет возможности показать CSM в верной пропорции с Землей и Луной. Для этого потребовалась бы площадь экрана *меньше пикселя*, что, по сути, невозможно. Да и какой в этом смысл? Поэтому мы вновь допускаем вольность в отношении масштабирования и делаем CSM достаточно большим, чтобы его можно было признать как космический корабль «Аполлон».

Во-вторых, мы не будем использовать для указателя CSM стандартное изображение, как это делали для других тел. Причина в том, что изображения не вращаются автоматически при повороте указателя, а нам нужно, чтобы CSM большую часть полета был направлен кормой вперед. Поэтому мы вынуждены создать для него настраиваемую фигуру.

Листинг 6.8 продолжает `main()`. Здесь мы рисуем базовые фигуры, такие как прямоугольники и треугольники. Затем совмещаем их в единое изображение, чтобы создать представление CSM.

Листинг 6.8. Создание настраиваемой фигуры для указателя CSM

apollo_8_free_return.py, часть 8

```
        csm = Shape('compound')
        cm = ((0, 30), (0, -30), (30, 0))
        csm.addcomponent(cm, 'white', 'white')
        sm = ((-60, 30), (0, 30), (0, -30), (-60, -30))
        csm.addcomponent(sm, 'white', 'black')
        nozzle = ((-55, 0), (-90, 20), (-90, -20))
        csm.addcomponent(nozzle, 'white', 'white')
        screen.register_shape('csm', csm)
```

Создаем переменную `csm` и вызываем `turtle`-класс `Shape`. Передаем ему `'compound'`, указывающий, что нужно создать фигуру из нескольких элементов. Первым будет командный модуль. Создаем переменную `cm` и присваиваем ее кортежу пар координат, в `turtle` известному как *polygon* (многоугольник). Эти координаты обозначают треугольник, как показано на рис. 6.12.

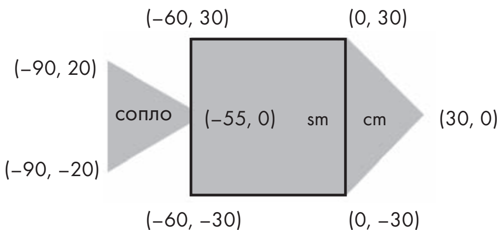


Рис. 6.12. Составной указатель CSM, на котором отмечены координаты для сопла, служебного (sm) и командного (cm) модулей

Добавляем этот треугольник к фигуре `csm` при помощи метода `addcomponent()`, вызываемого через точечную нотацию. Передаем ему переменную `cm`, цвет заполнения и цвет контура. Выбираем для этого белый, серебристый, серый или красный.

Повторяем этот общий процесс для прямоугольника, обозначающего служебный модуль. Устанавливаем цвет контура как `black`, чтобы отделить его от командного модуля (рис. 6.12).

Еще одним треугольником обозначаем сопло. Добавляем этот элемент и затем регистрируем новый составной указатель `csm` на экране. Передаем этому методу имя для фигуры и переменную, ссылающуюся на нее.

Создание CSM, запуск симуляции и вызов main()

Листинг 6.9 завершает функцию `main()`, инстанцируя указатель для CSM и вызывая цикл симуляции, который выполняет шаги по времени. Если программа работает в автономном режиме, то в завершение вызывается `main()`.

Листинг 6.9. Инстанцирование указателя CSM, вызов цикла симуляции и main()

apollo_8_free_return.py, часть 9

```
    ship = Body(1, (Ro_X, Ro_Y), Vec(Vo_X, Vo_Y), gravsys, 'csm')
    ship.shapesize(0.2)
    ship.color('white')
    ship.getscreen().tracer(1, 0)
    ship.setheading(90)

    gravsys.sim_loop()

if __name__ == '__main__':
    main()
```

Создаем указатель ship, который представляет CSM. Его начальная позиция — кортеж (x, y), помещающий CSM на временную орбиту прямо под указателем Земли. Я сначала аппроксимировал подходящую высоту для временной орбиты (R_0 на рис. 6.3), а затем подкорректировал ее через повторение симуляции.

Обратите внимание, что мы используем не фактические значения, а константы, присвоенные в начале программы. Суть в том, чтобы облегчить вам работу, если вы решите поэкспериментировать с этими значениями позднее.

Аргумент скорости (Vo_X, Vo_Y) представляет скорость CSM в момент, когда третья ступень «Сатурна» отключается при выходе на транслунную орбиту. Вся тяга направлена в сторону x, но земная гравитация приведет к мгновенному изменению траектории полета — вверх. Как и параметр R_0, значение скорости выбрано примерно и потом скорректировано в процессе симуляции. Обратите внимание, что скорость — это кортеж, вводимый при помощи вспомогательного класса Vec2D, который позволяет дальнейшим методам изменять ее значение посредством векторной арифметики.

Далее с помощью метода shapesize() устанавливаем размер указателя ship, после чего назначаем для траектории цвет white, чтобы он соответствовал цвету ship. Другие интересные варианты цветов — серебристый, серый и красный.

Контролируем обновление экрана с помощью методов getscreen() и tracer(), описанных в листинге 6.7, после чего устанавливаем направление корабля на 90 градусов, которые отсчитываются от условного востока на экране.

На этом создание объектов тел завершается. Теперь осталось только запустить цикл симуляции, используя метод sim_loop() объекта gravsys. Возвращаясь в глобальную область, заканчиваем программу кодом для ее выполнения в качестве импортируемого модуля или автономно.

Поскольку программа написана, нужно вручную закрыть окно Turtle Graphics. Если вы захотите, чтобы оно закрывалось автоматически, добавьте в последнюю строку `main()`такую команду:

```
screen.bye()
```

Выполнение симуляции

При первом запуске симуляции рисование траекторий движения тел будет отключено (рис. 6.13). CSM будет плавно поворачиваться, меняя ориентацию по мере приближения к Луне, а затем к Земле.

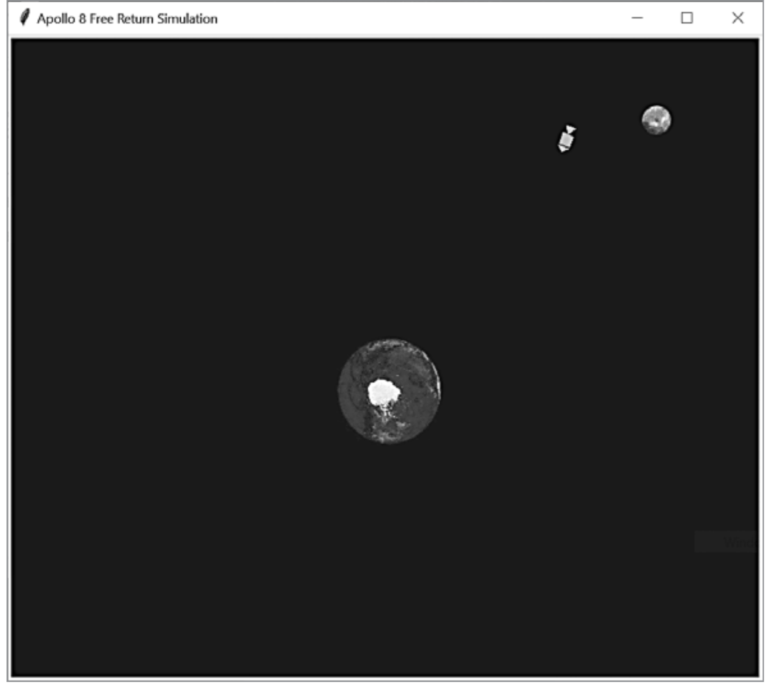


Рис. 6.13. Симуляция, выполняемая с отключенным рисованием траектории, по которой CSM приближается к Луне

Для отслеживания полета CSM перейдите к определениям класса `Body` и раскомментируйте строку:

```
self.pendown() # Раскомментируйте, чтобы рисовать путь позади объектов
```

Теперь на экране отобразится траектория возвращения корабля, напоминающая восьмерку (рис. 6.14).

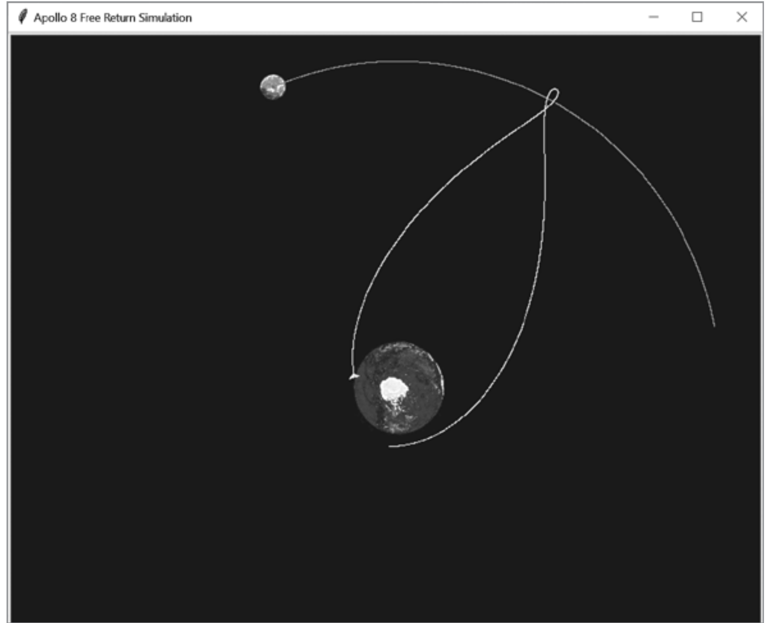


Рис. 6.14. Выполнение симуляции с отрисовкой траекторий движения, согласно которой CMS приводняется в Тихом океане

Можно также симулировать *гравитационный толчок*, иначе называемый *пертурбационным маневром*, установив переменную скорости `Vo_X` в значение между 520 и 540 и перезапустив симуляцию. В результате CSM пройдет за Луной и «присвоит» часть ее импульса, увеличив собственную скорость и отклонившись со своей траектории полета (рис. 6.15). Прощай, «Аполлон-8»!

Выполнив этот проект, вы должны усвоить, что космическое путешествие — это игра секунд и сантиметров. Если вы продолжите экспериментировать со значением переменной `Vo_X`, то обнаружите, что даже небольшие изменения могут загубить всю миссию. Если корабль не врежется в Луну, то войдет в земную атмосферу под излишне крутым углом или вообще промахнется мимо Земли.

В симуляциях хорошо то, что, если вы провалите миссию, то останетесь живы и сможете начать все сначала. В NASA прокручивают бесчисленное множество симуляций различных вариантов полета, находят наиболее эффективные маршруты, решают, что делать в случае неполадок, и многое другое.

Симуляции особенно важны для исследования внешней области Солнечной системы, где громадные расстояния делают невозможной коммуникацию

в реальном времени. Тайминг ключевых событий, таких как срабатывание тяговых двигателей, фотосъемка или сброс зондов, — все предварительно программируются на основе тщательных симуляций.

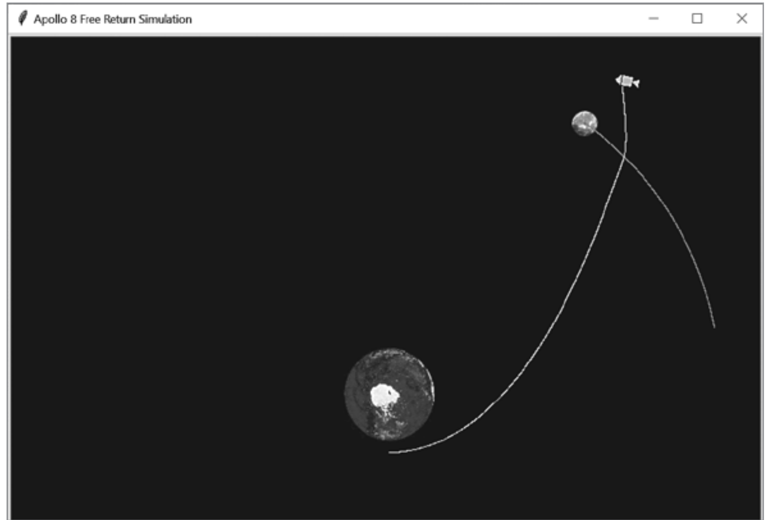


Рис. 6.15. Гравитационный маневр, полученный за счет значения Vo_X = 520

Итоги

В этой главе вы узнали, как использовать программу рисования `turtle`, включая создание настраиваемых указателей, а также научились применять Python для симулирования гравитации и решения знаменитой задачи трех тел.

Дополнительная литература

«Apollo 8: The Thrilling Story of the First Mission to the Moon»[1] (Henry Holt and Co., 2017), написанная Джеффри Клугером (Jeffrey Kluger), рассказывает об исторической миссии «Аполлона-8», начиная от ее малоперспективного начала и до «невообразимого триумфа».

Онлайн-поиск по запросу *«PBS Nova How Apollo 8 left Earth Orbit»* должен выдать короткий ролик, на котором показан маневр выхода «Аполлона-8» на транслунную орбиту — первый полет в истории, когда человеку удалось покинуть земную орбиту и долететь до другого небесного тела.

[1] Клугер Дж. «“Аполлон-8”. Захватывающая история первого полета к Луне».

В «NASA Voyager 1 & 2 Owner's Workshop Manual» (Haynes, 2015) авторы Кристофер Райли (Christopher Riley), Ричард Корфилд (Richard Corfield) и Филип Доллинг (Philip Dolling) интересно рассказывают о предыстории задачи трех тел и богатом вкладе Майкла Миновича в космические путешествия.

Страница Википедии Gravity assist (https://en.wikipedia.org/wiki/Gravity_assist) содержит много интересных анимаций и исторических примеров их применения, которые можно воссоздать с помощью симуляции «Аполлона-8».

«Chasing New Horizons: Inside the Epic First Mission to Pluto»[1] (Picador, 2018), написанная Аланом Стерном (Alan Stern) и Дэвидом Гринспуном (David Grinspoon), рассказывает о важности — и повсеместности — симуляций в миссиях NASA.

Практический проект: симуляция шаблона поисков

В главе 1 мы использовали теорему Байеса, чтобы помочь береговой охране найти потерянного в море моряка. Теперь используем `turtle`, чтобы спроектировать для вертолета шаблон поиска того же моряка. Предположим, что воздушные наблюдатели могут видеть на 20 пикселей, и сделаем расстояние между длинными отрезками пути равным 40 пикселей (рис. 6.16).

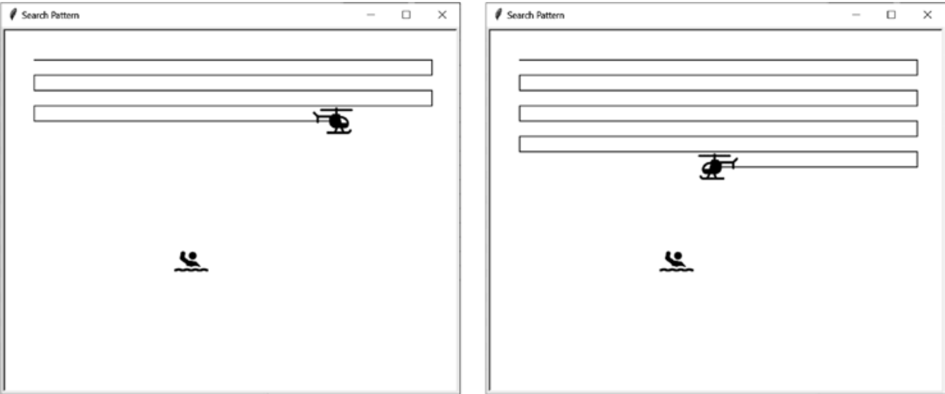


Рис. 6.16. Два скриншота из practice_search_pattern.py

[1] Стерн А., Гринспун Д. «За новыми горизонтами. Первый полет к Плутону».

Ради интереса добавьте указатель-вертолет так, чтобы при завершении каждого прохода над поверхностью он разворачивался в правильном направлении. Также добавьте указатель для моряка — в случайной позиции, реализуйте остановку симуляции, когда его обнаружат, и опубликуйте радостную новость на экране (рис. 6.17).

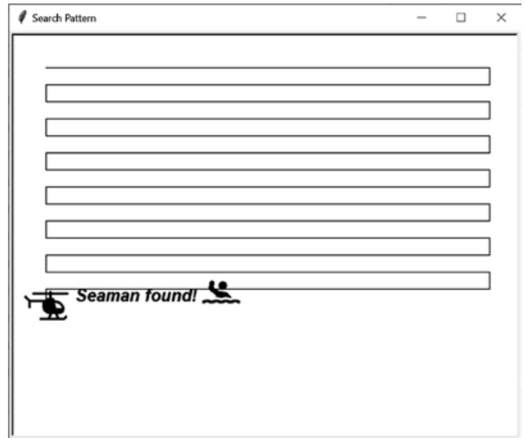


Рис. 6.17. Моряк замечен в practice_search_pattern.py (Seaman found – моряк найден!)

Решение под названием practice_search_pattern.py можно найти в приложении к книге. Его цифровую версию, а также указатели для вертолета и моряка я положил в каталог Chapter_6, который можно скачать с сайта книги.

Практический проект: запусти меня!

Перепишите Apollo_8_free_return.py, чтобы движущаяся Луна приближалась к находящемуся в покое CSM, заставляла его начать движение, а затем запускала его вверх и в сторону. Ради интереса сориентируйте указатель CSM так, чтобы он всегда смотрел в направлении полета, как если бы ускорялся самостоятельно (рис. 6.18).

Решение под названием practice_grav_assist_stationary.py находится в приложении, а также доступно для скачивания с https://nostarch.com/real-world-python/.

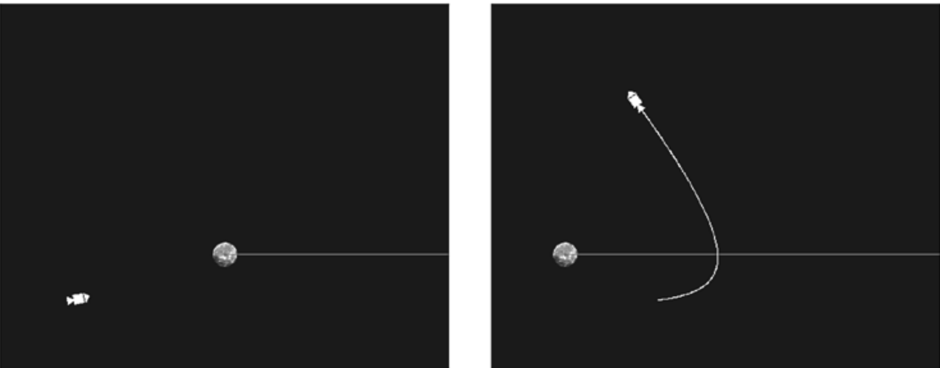


Рис. 6.18. Луна приближается к неподвижному CSM (*слева*), а затем запускает его к звездам (*справа*)

Практический проект: останови меня!

Перепишите Apollo_8_free_return.py, чтобы орбиты Луны и CSM пересекались. Пусть CSM проходит перед Луной, и ее гравитация замедляет его движение почти до нуля, а также меняет направление полета на угол около 90 градусов. Как и в предыдущем проекте, пусть CSM смотрит в направлении своего полета (рис. 6.19).

Рис. 6.19. Орбиты Луны и CSM пересекаются, при этом Луна замедляет и поворачивает CSM

Решение под названием practice_grav_assist_intersecting.py доступно в приложении, а также на сайте книги https://nostarch.com/real-world-python/.

Усложняем проект: симуляция в истинном масштабе

Перепишите Apollo_8_free_return.py так, чтобы Земля, Луна и расстояние были показаны в верном масштабе, как на рис. 6.10. Вместо изображений небесных тел используйте цветные круги, а CSM сделайте невидимым (просто рисуйте его траекторию). Информация из табл. 6.2 поможет вам определить относительные размеры и нужные расстояния.

Таблица 6.2. Параметры длин для системы Земля — Луна

Радиус Земли	6371 км
Радиус Луны	1737 км
Расстояние от Земли до Луны	356 700 км*

* Максимальное приближение во время миссии «Аполлон-8» в декабре 1968 года.

Усложняем проект: реальный «Аполлон-8»

Перепишите Apollo_8_free_return.py так, чтобы программа симулировала всю миссию «Аполлон-8», а не только траекторию свободного возвращения. CSM должен облететь Луну 10 раз и лишь затем вернуться на Землю.

7

Выбор мест высадки на Марсе

Посадка космического аппарата на Марсе — невероятно сложная и опасная процедура. Никто не хочет терять зонд ценой в миллиард долларов, поэтому инженеры должны максимально обезопасить такую операцию. Они могут годами изучать спутниковые снимки в поиске наиболее безопасных мест посадки, удовлетворяющих целям миссии. Исследовать при этом приходится огромные пространства поверхности, так как площадь суши на Марсе практически равна земной.

Для анализа настолько обширной области требуется помощь вычислительных машин. В этой главе мы используем Python и гордость Jet Propulsion Laboratory — карту лазерного высотомера MOLA, чтобы отобрать и ранжировать предполагаемые места посадки модуля. Чтобы скачать карту и извлечь из нее нужную информацию, мы используем Python Imaging Library, OpenCV, `tkinter` и `NumPy`.

Посадка на Марс

Есть много способов посадить на Марс зонд, включая использование парашютов, шаров, ракет с тормозными устройствами и реактивных летательных аппаратов. Независимо от выбранного метода большая часть процесса посадки выполняется согласно одним и тем же базовым правилам безопасности.

Первое правило — садиться в низинных областях. Зонд может войти в марсианскую атмосферу со скоростью до 27 000 км/ч, и чтобы достаточно замедлить

его для плавной посадки, требуется плотная атмосфера. Но на Марсе она разреженная — примерно 1 % от земной. Чтобы задействовать ее по полной при торможении, необходимо выбирать низины, где воздух плотнее, но полет до этих локаций требует больше времени.

Если мы сажаем не какой-то специализированный зонд, например разработанный для исследования полярных областей, то лучше всего целиться в область экватора. Здесь достаточно света для питания солнечных батарей зонда, а температуры весьма высокие для сохранения функциональности его чувствительного оборудования.

При этом нужно избегать областей суши, покрытых валунами, которые могут уничтожить зонд, не дать раскрыться его панелям, заблокировать механическую руку или воспрепятствовать попаданию солнечных лучей. По схожим причинам следует избегать областей с крутыми склонами, например краев кратеров. С позиции безопасности чем более плоская поверхность, тем лучше.

Еще одна сложность при посадке на Марс — невозможно прицелиться точно. Трудно пролететь 50 миллионов километров или даже более, войти в атмосферу и выполнить посадку ровно в намеченном месте. Неточности в межпланетной навигации наряду с изменчивостью марсианской атмосферы затрудняют точность посадки.

Следовательно, в NASA прокручивают большое количество компьютерных симуляций для каждого места посадки. Каждый прогон симуляции дает координату, а разброс точек, получающихся из тысяч таких прогонов, формирует эллиптическую фигуру с длинной осью, параллельной траектории полета зонда. Эти *посадочные эллипсы* могут получаться довольно большими (рис. 7.1), хотя с каждой новой миссией точность повышается.

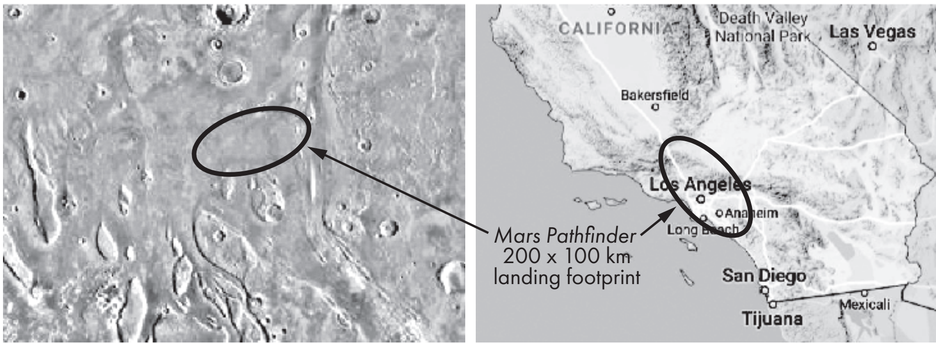


Рис. 7.1. Сравнение масштабов посадочной области Mars Pathfinder в 1997 году (*слева*) и в Южной Калифорнии (*справа*)

Посадочный модуль *InSight*, совершивший посадку на Марсе в 2018 году, целился в область площадью всего 130 × 27 км. Вероятность посадки внутри этого эллипса составляла около 99 %.

Карта MOLA

Для определения удачных точек посадки потребуется карта Марса. В период между 1997 и 2001 годами установленный на борту *Mars Global Surveyor (MGS)* прибор сделал 600 миллионов измерений отражения лазерного луча от поверхности планеты. На основе этих данных исследователи под руководством Мари Зубер (Maria Zuber) и Дэвида Смита (David Smith) создали детализированную глобальную топографическую карту, названную MOLA (рис. 7.2).

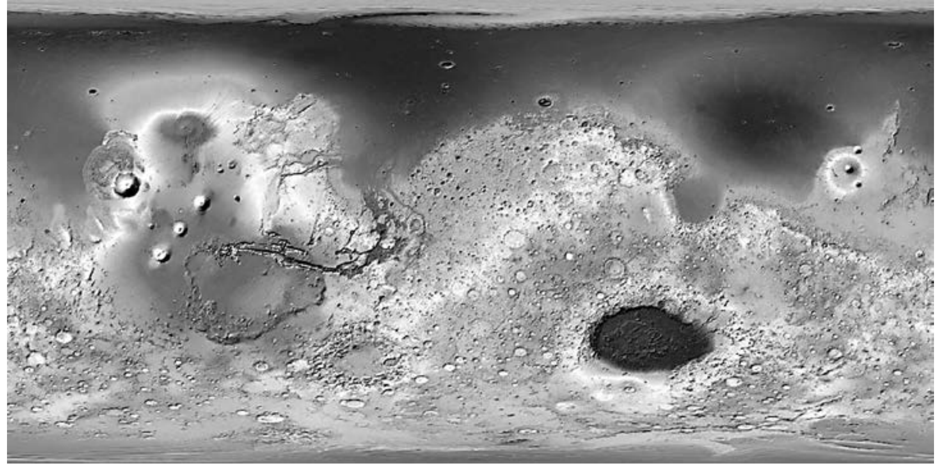


Рис. 7.2. Затененная карта рельефа Марса MOLA

Чтобы увидеть эффектную цветную версию MOLA вместе с описанием, перейдите на страницу Википедии *Mars Global Surveyor*. Синим на этой карте обозначены места, где миллиарды лет назад, возможно, существовали океаны и моря. Их распределение основано на комбинации высоты и признаков поверхности, напоминающих древние береговые линии.

Точность лазерных измерений, на основе которых составлялась MOLA, для вертикального направления равняется примерно от 3 до 13 м, а для горизонтального — около 100 м. Разрешающая способность изображения составляет 463 метров на пиксель. Сама по себе карта не содержит деталей, необходимых для безопасного выбора посадочного эллипса, но отлично подойдет для предварительной работы, которую мы и собираемся проделать.

Проект #10. Выбор посадочных мест на Марсе

Представьте, что в качестве интерна вы работаете в NASA над проектом «Орфей». Этот проект посвящен прослушиванию марсотрясений и изучению внутренней структуры планеты, наподобие миссии *InSight* 2018 года. Поскольку цель «Орфея» — изучение внутреннего строения Марса, то интересные элементы поверхности планеты особой роли не играют. В первую очередь важна безопасность, что делает этот проект мечтой любого инженера.

Наша задача — найти не менее десятка прямоугольных областей, из которых специалисты NASA смогут отобрать варианты посадочных площадок в форме эллипса. Ваш наставник поясняет, что искомые области должны представлять прямоугольники длиной 670 км (В–З) и шириной 335 км (С–Ю). Исходя из требований безопасности, эти локации должны располагаться по обе стороны экватора между 30° северной и 30° южной широты, находиться в низинной части и быть максимально ровными и плоскими.

ЗАДАЧА

Написать программу Python, которая использует карту MOLA для выбора 20 наиболее безопасных областей площадью 670 х 335 км возле марсианского экватора, из которых можно будет отобрать посадочные площадки эллиптической формы для модуля «Орфей».

Стратегия

Сначала нам надо найти способ разделить цифровую карту MOLA на прямоугольные области и собрать статистические данные о том, на какой высоте они располагаются и насколько ровная их поверхность. Это означает, что мы будем работать с пикселями, поэтому потребуются инструменты для обработки изображений. А поскольку NASA всегда стремится к экономии, то и использовать придется бесплатные открытые библиотеки, такие как OpenCV, Python Imaging Library (PIL), `tkinter` и `NumPy`. Обзор и инструкции об установке OpenCV и `NumPy` ищите в разделе «Установка библиотек Python» на с. 31, а о PIL — в разделе «Модули Word Cloud и PIL» на с. 102. Что же касается `tkinter`, то этот модуль изначально включен в Python.

Чтобы учесть перепад уровня высот для каждой области, можно просто вычислить среднюю высоту. Для измерения степени пересеченности поверхности в заданном масштабе можно использовать разные варианты, в том числе весьма замысловатые. Помимо определения того, насколько поверхность ровная, на основе данных о высоте, можно поискать дифференциальное затенение

на стереоизображениях; количество рассеивания в радиолокационных, лазерных и микроволновых отражениях; термические вариации в инфракрасных изображениях и т. д. Многие оценки основаны на утомительном анализе местности вдоль *трансектов*. Трансекты — это линии, нарисованные на поверхности планеты, вдоль которых измеряют и тщательно исследуют перепады высот. Ну а поскольку вы все же не интерн, которого взяли на лето и которого интересует лишь как поскорее освободиться от трехмесячной практики, то вам нужно хорошо решить задачу. Постараемся все упростить — для этого используем два типичных измерения, которые применимы к каждой прямоугольной области: стандартное отклонение и значение перепада высот.

Стандартное отклонение (standard deviation, StD), также называемое физиками *среднеквадратическим*, представляет меру распространения множества чисел. Низкое стандартное отклонение указывает, что значения во множестве близки к среднему. Высокое же свидетельствует о том, что они распределены по более обширному диапазону. Область карты с низким стандартным отклонением высоты означает, что эта область плоская с перепадами, немного отличающимися от среднего значения высоты.

Технически стандартное отклонение для набора образцов — это квадратный корень из усредненного квадрата отклонений от среднего значения. Представляется же оно следующей формулой:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(h_i - h_0)^2}.$$

Здесь σ — это стандартное отклонение, N — количество образцов, h_i — текущий образец высоты, а h_0 — среднее по всем высотам.

Высота неровностей профиля (peak-to-valley, PV) — это разница в высоте между самой высокой и самой низкой точками поверхности, то есть максимальное изменение высоты поверхности. Это важный показатель, так как поверхность может обладать относительно низким, предполагающим ровную поверхность, стандартным отклонением, но при этом быть небезопасной, как показано на рис. 7.3.

Показатели стандартного отклонения и высоты неровностей профиля можно использовать в качестве сравнительных метрик. Для каждой прямоугольной области мы ищем наименьшие значения каждой из этих метрик. А так как каждая запись несколько отличается, то мы отыщем 20 лучших областей для каждой из этих метрик, а затем выберем только те области, для которых метрики пересекаются. Это позволит найти наиболее удачные прямоугольные площадки.

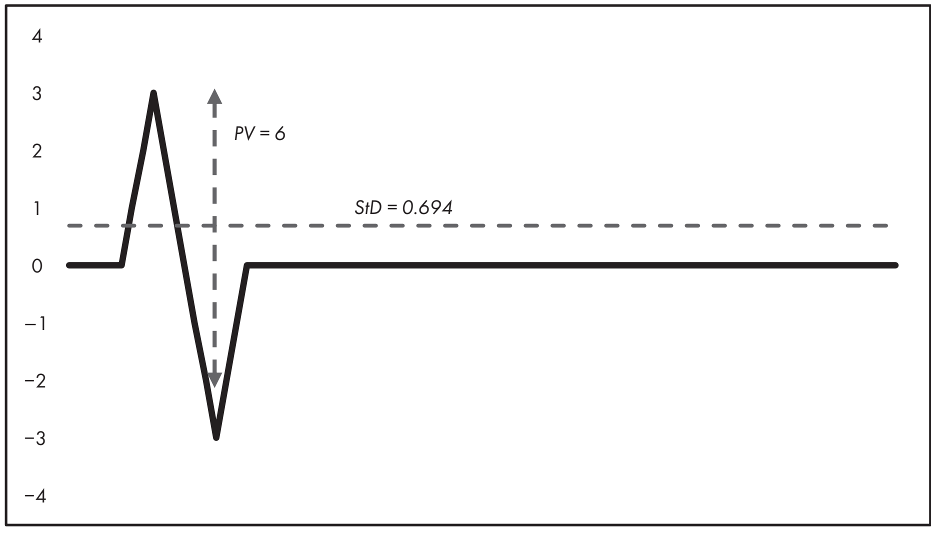


Рис. 7.3. Профиль поверхности (черная линия) с указанием стандартного отклонения (StD) и высоты неровностей профиля (PV)

Код для выбора мест посадки

В программе site_selector.py используются полутоновое изображение карты (рис. 7.4) для выбора прямоугольных посадочных областей и затененная цветная

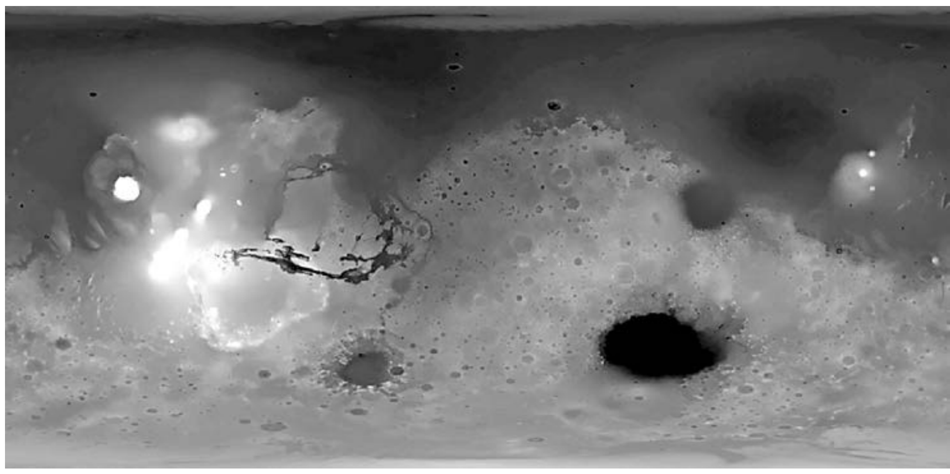


Рис. 7.4. Цифровая модель высот MGS MOLA с разрешением 463 метра на пиксель v2 (mola_1024x501.png)

карта (см. рис. 7.2) для их отметки. На полутоновом изображении высота представлена одним каналом, поэтому его проще использовать, чем трехканальную RGB-картинку.

Программа, полутоновое изображение (mola_1024×501.png) и цветное изображение (mola_color_1024×506.png) находятся в каталоге Chapter_7, доступном для скачивания с https://nostarch.com/real-world-python/. Разместите эти файлы в одном каталоге и не переименовывайте.

ПРИМЕЧАНИЕ

Карта MOLA доступна в различных размерах и разрешениях. Мы сейчас используем самую маленькую с целью сокращения времени скачивания и выполнения.

Импорт модулей и присваивание вводимых пользователем констант

Код листинга 7.1 импортирует модули и присваивает константы, представляющие вводимые пользователем параметры: имена изображений, размеры прямоугольных областей, максимальный порог высоты и количество предлагаемых к рассмотрению прямоугольников.

Листинг 7.1. Импорт модулей и присваивание вводимых пользователем констант

site_selector.py, часть 1

```
import tkinter as tk
from PIL import Image, ImageTk
import numpy as np
import cv2 as cv

# Константы: пользовательский ввод:
IMG_GRAY = cv.imread('mola_1024x501.png', cv.IMREAD_GRAYSCALE)
IMG_COLOR = cv.imread('mola_color_1024x506.png')
RECT_WIDTH_KM = 670
RECT_HT_KM = 335
MAX_ELEV_LIMIT = 55
NUM_CANDIDATES = 20
MARS_CIRCUM = 21344
```

Начинаем с импорта модуля `tkinter`. Это предустановленная в Python библиотека GUI для разработки десктопных приложений. С ее помощью мы будем создавать заключительное отображение: окно с цветной картой MOLA вверху и текстовым описанием отмеченных прямоугольников внизу. На большинстве машин с Windows, macOS и Linux `tkinter` уже установлен. Если у вас этой библиотеки нет или нужна последняя версия, то можно скачать и установить ее со страницы https://www.activestate.com/. Онлайн-документация для этого модуля находится по адресу https://docs.python.org/3/library/tk.html.

Далее импортируем модули `Image` и `ImageTK` из Python Imaging Library. `Image` предоставляет класс, отображающий картинку PIL. Кроме того, он содержит функции, в том числе необходимые для загрузки изображений из файлов и создания новых изображений. Модуль `ImageTK` обеспечивает поддержку для создания и изменения объектов `BitmapImage` библиотеки `tkinter` из изображений PIL. Опять же, мы используем их в конце программы, чтобы разместить в итоговом окне цветную карту и описания. В завершение импортируем библиотеки `NumPy` и OpenCV.

Теперь присваиваем представляющие пользовательский ввод константы, которые по ходу выполнения программы изменяться не будут. Сначала с помощью метода OpenCV `imread()` скачиваем полутоновое изображение MOLA. Обратите внимание, что нужно использовать флаг `cv.IMREAD_GRAYSCALE`, поскольку этот метод по умолчанию скачивает изображения в цвете. Повторяем тот же код без флага, чтобы скачать уже цветной вариант. Затем добавляем константы для размера прямоугольников. В следующем листинге мы преобразуем эти размеры в пиксели для использования совместно с картой.

Теперь, чтобы наши прямоугольники гарантированно попали в низинные ровные области, нужно ограничить поиск плоскими участками с малым количеством кратеров. Считается, что эти области — дно древних океанов. Таким образом, необходимо установить верхний порог высоты до полутонового значения 55, которое примерно соответствует областям, считающимся древними береговыми линиями (рис. 7.5).

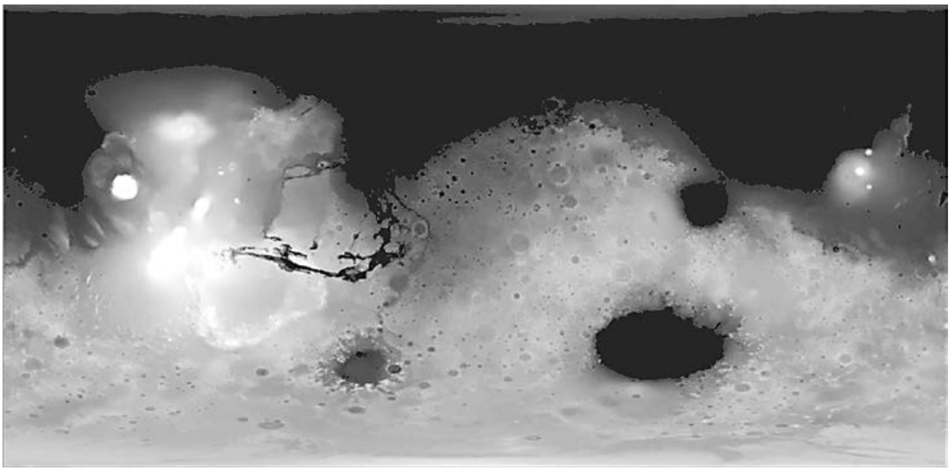


Рис. 7.5. Карта MOLA, где области со значениями пикселей <= 55 закрашены черным для выделения древних марсианских океанов

Теперь в переменной NUM_CANDIDATES указываем количество прямоугольников для отображения. Позднее мы выберем их из упорядоченного списка статистических данных о прямоугольниках. Завершаем код назначением константы, которая будет хранить протяженность окружности Марса в километрах. Ее мы используем позже для определения количества пикселей на километр.

Присваивание выводимых констант и создание объекта экрана

В листинге выполняется присваивание констант, выводимых из других констант. Эти значения будут обновляться автоматически при изменении пользователем предыдущих констант, например для тестирования размеров прямоугольников или границ высоты. Оканчивается этот листинг созданием tkinter-объектов screen и canvas для итогового отображения.

Листинг 7.2. Присваивание выводимых констант и настройка экрана tkinter

site_selector.py, part 2

```
  # Константы: выводимые:
  IMG_HT, IMG_WIDTH = IMG_GRAY.shape
  PIXELS_PER_KM = IMG_WIDTH / MARS_CIRCUM
  RECT_WIDTH = int(PIXELS_PER_KM * RECT_WIDTH_KM)
  RECT_HT = int(PIXELS_PER_KM * RECT_HT_KM)
❶ LAT_30_N = int(IMG_HT / 3)
  LAT_30_S = LAT_30_N * 2
  STEP_X = int(RECT_WIDTH / 2)
  STEP_Y = int(RECT_HT / 2)

❷ screen = tk.Tk()
  canvas = tk.Canvas(screen, width=IMG_WIDTH, height=IMG_HT + 130)
```

Распаковываем высоту и ширину изображения с помощью атрибута shape. OpenCV сохраняет изображения как nd-массивы NumPy, являющиеся n-мерными массивами или таблицами элементов одного типа. Для массива изображений shape является кортежем из количества рядов, столбцов и каналов. Высота представляет количество пиксельных строк в изображении, а ширина — количество пиксельных столбцов. Каналы представляют число элементов, используемых для выражения каждого пикселя (например, красный, зеленый и синий). Для полутоновых изображений с одним каналом shape является просто кортежем из высоты и ширины области.

Чтобы преобразовать измерение прямоугольных областей из километров в пиксели, нужно знать, сколько пикселей в каждом километре. Поэтому мы делим ширину изображения на длину окружности, получая тем самым соотношение пикселей на километр в области экватора. Затем преобразуем ширину и высоту в пиксели. Эти результаты понадобятся, чтобы вывести значения для

квантования индекса, поэтому необходимо, чтобы они были целыми числами, для чего используется `int()`. Значение этих констант теперь должно быть 32 и 16 соответственно.

Нам нужно ограничить поиск наиболее теплыми и солнечными участками, которые располагаются с двух сторон от экватора — между 30° северной и 30° южной широты (рис. 7.6). Можно сказать, что этот регион соответствует земным тропикам.

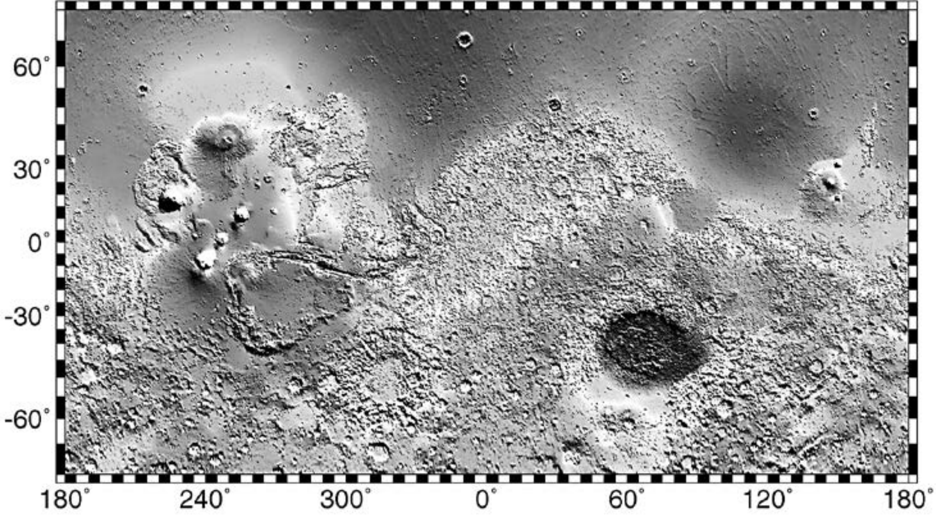


Рис. 7.6. Широта (ось y) и долгота (ось x) на Марсе

Значения широты начинаются от 0° у экватора и заканчиваются на 90° у полюсов. Чтобы найти 30° северной широты, нужно просто разделить высоту изображения на 3 ❶. Чтобы получить 30° для южной широты, нужно удвоить число пикселей, которое потребовалось для нахождения предыдущего значения для севера.

Ограничение поиска экваториальным регионом имеет положительный побочный эффект. Используемая карта MOLA основана на *цилиндрической проекции*, применяемой для отображения карты с глобуса на плоскость. В результате такого переноса сходящиеся линии долготы становятся параллельными, что сильно искажает детали в районе полюсов. Вы могли заметить это по настенным картам Земли, где Гренландия выглядит как континент, а Антарктида невероятно огромная (рис. 7.7).

К счастью, в области экватора это искажение минимизируется, и уже не нужно учитывать его в размерах прямоугольников. Убедиться в этом можно, проверив

форму кратеров на карте MOLA. До тех пор пока они остаются аккуратными и круглыми (а не овальными), вызываемые проекцией эффекты можно игнорировать.

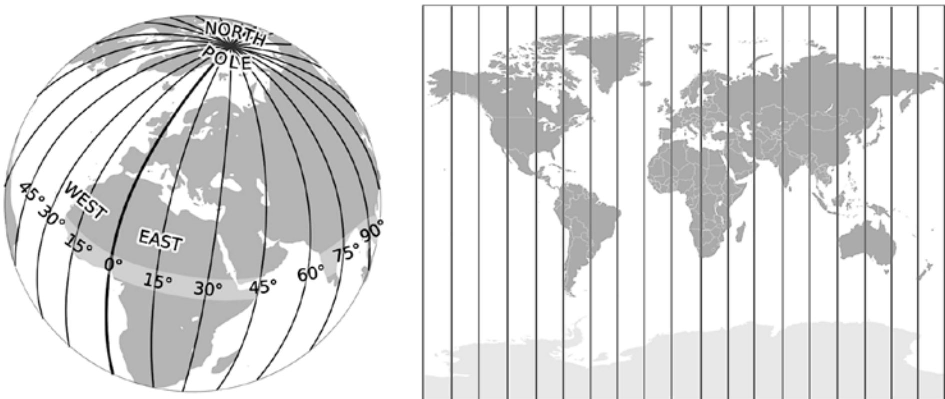


Рис. 7.7. Выпрямление линий долготы до параллельных приводит к искажению деталей в районе полюсов

Далее нужно поделить карту на прямоугольные области. Логично начать с верхнего левого угла, расположенного под линией 30° северной широты (рис. 7.8).

Рис. 7.8. Расположение первого из прямоугольников

Программа нарисует этот прямоугольник, пронумерует его и вычислит внутреннюю статистику высот. Затем она сместит прямоугольник на восток и повторит процесс. Расстояние каждого очередного смещения определяется константами STEP_X и STEP_Y и зависит от так называемого *алиасинга*.

Алиасинг — это проблема разрешения. Он возникает, когда берется недостаточное количество образцов для определения всех важных деталей поверхности в некой области. Это может привести к тому, что мы «пропустим» такую деталь, как кратер, и не сможем его распознать. К примеру, на рис. 7.9A между двумя кратерами наблюдается подходящий гладкий посадочный эллипс. Однако, как мы видим, на рис. 7.9B этому эллипсу не соответствует ни одна прямоугольная область. Оба соседних прямоугольника частично включают края кратеров. В результате ни одна из очерченных ими областей не содержит подходящего посадочного эллипса, несмотря на то что в их смежной области он есть.

При таком порядке расположения прямоугольников эллипс на рис. 7.9A попадает под эффект *алиасинга*. Если же сместить каждый прямоугольник на половину его ширины, как на рис. 7.9C, то эта ровная область будет правильно отобрана и распознана.

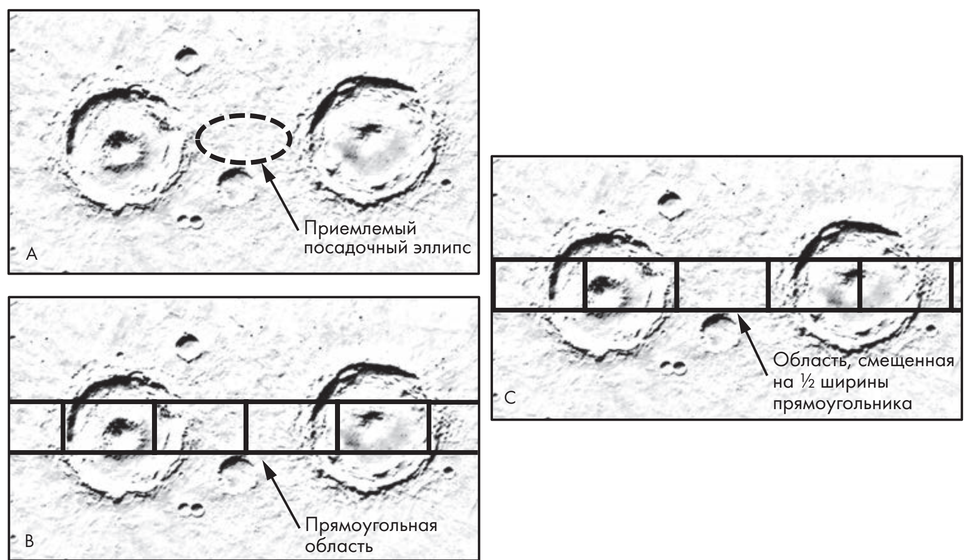


Рис. 7.9. Пример алиасинга, вызванного расположением прямоугольников

Общим правилом для предотвращения эффектов алиасинга является определение шага меньше или равным половине ширины наименьшей детали, которую

требуется определить. Для данного проекта мы используем половину ширины прямоугольника, чтобы не усложнять процесс отображения.

Теперь пора заглянуть вперед и разобрать заключительный вариант карты. Создаем экземпляр screen класса Tk() модуля tkinter ❷. Приложение tkinter является оболочкой Python для набора инструментов GUI под названием Tk, изначально написанным на языке TCL. Окно screen ему требуется для связи с внутренним интерпретатором tcl/tk, который переводит команды tkinter в команды tcl/tk.

Далее создаем tkinter-объект canvas. Это область для рисования прямоугольников, спроектированная для сложных графических макетов, текста, виджетов и фреймов. Передаем ей объект screen, устанавливаем ее ширину равной изображению MOLA, а высоту — равной изображению MOLA плюс 130. В дополнительной области под картинкой мы поместим текст с обобщающей статистикой отображаемых прямоугольников.

Обычно только что описанный код tkinter помещается в *конце* программ, а не в начале. Я же решил разместить его вверху, чтобы упростить восприятие сопутствующих пояснений. Также можно встроить его в функцию, выполняющую финальное отображение. Однако это может вызвать проблемы у пользователей macOS. Для версии macOS 10.6 или более поздних предоставляемый Apple набор инструментов Tcl/Tk 8.5 имеет серьезные баги, которые могут вызвать падение приложения (подробнее — по ссылке https://www.python.org/download/mac/tcltk/).

Определение и инициализация класса Search

В листинге 7.3 определяем класс, который мы используем для поиска подходящих прямоугольных областей. Следом идет определение метода инициализации _init_(), используемого для инстанцирования новых объектов. Чтобы вкратце узнать об ООП, обратитесь к разделу «Определение класса Search» на с. 36, где мы также определяли класс поиска.

Листинг 7.3. Определение класса Search и метода _init_()

site_selector.py, часть 3

```
class Search():
    """Считывание изображения и определение посадочных прямоугольников
    на основе входных критериев"""

    def __init__(self, name):
        self.name = name
      ❶ self.rect_coords = {}
        self.rect_means = {}
        self.rect_ptps = {}
        self.rect_stds = {}
```

```
    ❷ self.ptp_filtered = []
      self.std_filtered = []
      self.high_graded_rects = []
```

Определяем класс `Search`. Далее прописываем метод `_init_()`, используемый для создания новых объектов. Параметр `name` позволит давать собственное имя каждому создаваемому в функции `main()` объекту.

Теперь можно присваивать атрибуты. Начинаем со связывания имени объекта с аргументами, которые будем предоставлять при его создании. Далее присваиваем пустые словари для хранения важных статистических данных для каждого прямоугольника (rect) ❶. К ним относятся координаты его угловых точек, средняя высота (mean elevation), высота неровностей профиля (peak-to-valley) и стандартное отклонение (standard deviation). В качестве ключа во всех словарях будут использоваться последовательные числа, начиная с 1. Эти данные нужно отфильтровать, чтобы найти наименьшие значения, поэтому создаем два пустых списка для их хранения ❷. Обратите внимание, что для представления статистики высоты неровностей профиля (peak-to-valley) я использую выражение `ptp`, а не `ptv`. Это необходимо для согласования со встроенным в `NumPy` методом *peak-to-peak*, занимающимся ее вычислением.

В конце программы мы поместим прямоугольники, которые одновременно встречаются в упорядоченных списках стандартного отклонения и высот неровностей профиля, в новый список `high_graded_rects`. Этот список будет содержать номера прямоугольников с наименьшими общими показателями. Эти прямоугольники окажутся лучшими местами для посадочных эллипсов.

Вычисление статистик прямоугольников

Продолжая класс `Search`, код листинга 7.4 определяет метод, который вычисляет статистические показатели прямоугольника, добавляет их в соответствующие словари, а затем переходит к следующему прямоугольнику и повторяет процесс. Этот метод учитывает ограничение высоты, используя для заполнения словарей только прямоугольники из областей, расположенных в низинах.

Листинг 7.4. Вычисление статистик прямоугольника и переход к следующему

site_selector.py, часть 4

```
    def run_rect_stats(self):
        """Определение прямоугольных областей поиска и расчет внутренних
        статистических данных"""
        ul_x, ul_y = 0, LAT_30_N
        lr_x, lr_y = RECT_WIDTH, LAT_30_N + RECT_HT
        rect_num = 1
```

```
while True:
  ❶ rect_img = IMG_GRAY[ul_y : lr_y, ul_x : lr_x]
    self.rect_coords[rect_num] = [ul_x, ul_y, lr_x, lr_y]
    if np.mean(rect_img) <= MAX_ELEV_LIMIT:
        self.rect_means[rect_num] = np.mean(rect_img)
        self.rect_ptps[rect_num] = np.ptp(rect_img)
        self.rect_stds[rect_num] = np.std(rect_img)
    rect_num += 1

    ul_x += STEP_X
    lr_x = ul_x + RECT_WIDTH
  ❷ if lr_x > IMG_WIDTH:
        ul_x = 0
        ul_y += STEP_Y
        lr_x = RECT_WIDTH
        lr_y += STEP_Y
   ❸ if lr_y > LAT_30_S + STEP_Y:
        break
```

Определяем метод `run_rect_stats()`, получающий в качестве аргумента `self`. Далее присваиваем локальные переменные для верхнего левого и нижнего правого углов каждого прямоугольника. Инициализируем их с помощью комбинации координат и констант. Таким образом, мы поместим первый прямоугольник слева на изображении, прижав его верхнюю сторону к линии 30° северной широты.

Для отслеживания прямоугольников мы их именуем, начиная с 1. Эти числа будут служить ключами в словарях, используемых для записи координат и статистик. С их помощью мы также определим прямоугольники на карте, как было показано ранее на рис. 7.8.

Теперь начинаем цикл `while`, который автоматизирует процесс перемещения прямоугольников и запись статистических данных. Этот цикл будет выполняться, пока более половины прямоугольника не выступит за линию 30° южной широты, после чего произойдет `break`.

Как уже упоминалось, OpenCV сохраняет изображения как массивы `NumPy`. Чтобы вычислить статистические данные внутри активного прямоугольника, а не всего изображения, мы с помощью обычного разделения создаем подмассив ❶. Называем его `rect_img`. Далее добавляем номер прямоугольника и его координаты в словарь `rect_coords`. Нам нужно сохранить запись этих координат для специалистов NASA, которые впоследствии будут исследовать их более детально.

Теперь при помощи условного выражения проверим, соответствует ли текущий прямоугольник диапазону максимальной высоты, указанной для проекта. В этом же выражении с помощью `NumPy` вычисляем среднюю высоту для подмассива `rect_img`.

Если прямоугольник проходит проверку на высоту, то заполняем три словаря значениями координат, а также данными о высотах профиля и стандартном отклонении. Обратите внимание, что эти вычисления можно выполнить как часть процесса, используя `np.ptp` для высот и `np.std` для стандартного отклонения.

Далее увеличиваем переменную `rect_num` на `1` и смещаем прямоугольник. Сдвигаем верхнюю левую координату *x* на размер шага, а затем нижнюю правую координату *x* на ширину прямоугольника. Нам не нужно, чтобы он выдвинулся за правую границу изображения, поэтому проверяем, чтобы `lr_x` не превышала ширину изображения ❷. Если она окажется больше, то устанавливаем верхнюю левую координату *x* на `0`, чтобы переместить прямоугольник к начальной позиции у левого края экрана. Затем сдвигаем его координаты *y* вниз, чтобы новый ряд прямоугольников располагался под предыдущим. Если нижняя граница этого ряда окажется ниже 30° южной широты более чем на половину высоты прямоугольника, значит, исследование области поиска закончено и цикл можно завершать ❸.

Между 30° северной и южной широты изображение с обеих сторон окружено относительно гористой местностью с большим количеством кратеров, которая не подходит для посадки аппаратов (рис. 7.6). Ввиду этого можно игнорировать последний шаг, смещающий прямоугольник на половину его ширины. В противном случае потребуется добавить код, который будет делать перенос прямоугольника с одной стороны изображения на противоположную и вычислять статистику для каждой части. Мы более подробно рассмотрим эту ситуацию в конце главы.

ПРИМЕЧАНИЕ

Если вы нарисуете что-то на изображении, например, прямоугольника, рисунок станет частью этого изображения. Измененные пиксели будут включены в любой выполняемый анализ NumPy, поэтому все статистические данные необходимо вычислять до внесения описания.

Проверка местоположений прямоугольников

Листинг 7.5 по-прежнему продолжает класс `Search`. Здесь мы определяем метод, выполняющий контроль качества. Он выводит координаты (coords) всех прямоугольников, после чего рисует их на карте MOLA. Это позволит нам убедиться, что область поиска мы охватили полностью и прямоугольники имеют ожидаемые размеры.

Листинг 7.5. Рисование всех прямоугольников на карте MOLA в качестве этапа контроля качества

site_selector.py, часть 5

```
    def draw_qc_rects(self):
            """Нарисовать перекрывающиеся прямоугольники поиска на изображении
            в качестве проверки."""
```

```
        img_copy = IMG_GRAY.copy()
        rects_sorted = sorted(self.rect_coords.items(), key=lambda x: x[0])
        print("\nRect Number and Corner Coordinates
               (ul_x, ul_y, lr_x, lr_y):")
        for k, v in rects_sorted:
            print("rect: {}, coords: {}".format(k, v))
            cv.rectangle(img_copy,
                         (self.rect_coords[k][0], self.rect_coords[k][1]),
                         (self.rect_coords[k][2], self.rect_coords[k][3]),
                         (255, 0, 0), 1)
        cv.imshow('QC Rects {}'.format(self.name), img_copy)
        cv.waitKey(3000)
        cv.destroyAllWindows()
```

Начинаем с определения метода для рисования на изображении прямоугольников. Все, что мы рисуем на изображении в OpenCV, становится частью этого изображения, поэтому сначала сделайте его копию в локальном пространстве.

Нам нужно предоставить NASA порядковый номер и координаты каждого прямоугольника. Для вывода этих данных в числовом порядке мы их упорядочиваем в словаре `rect_coords`, используя лямбда-функцию. Если ранее такие функции вам использовать не доводилось, то краткое их описание вы найдете на с. 148 в главе 5.

Выводим заголовок для этого списка, а затем запускаем цикл `for`, перебирающий ключи и значения в упорядоченном словаре. Ключ представляет номер прямоугольника, а значение — список его координат, как показано в следующем выводе:

```
Номер прямоугольника и координаты углов (ul_x, ul_y, lr_x, lr_y):
rect: 1, coords: [0, 167, 32, 183]
rect: 2, coords: [16, 167, 48, 183]

--snip--

rect: 1259, coords: [976, 319, 1008, 335]
rect: 1260, coords: [992, 319, 1024, 335]
```

Для рисования прямоугольников на изображении используем метод OpenCV `rectangle()`. Передаем ему изображение, координаты прямоугольника, цвет и толщину линии. К координатам обращаемся напрямую через словарь `rect_coords`, используя ключ и индекс списка (0 = верхняя левая координата x, 1 = верхняя левая y, 2 = нижняя правая x, 3 = нижняя правая y).

Для показа изображения вызываем метод OpenCV `imshow()`, передавая ему имя окна и переменную изображения. Прямоугольники должны покрывать поверхность Марса в области вокруг экватора (рис. 7.10). Оставляем окно активным на 3 секунды, затем закрываем.

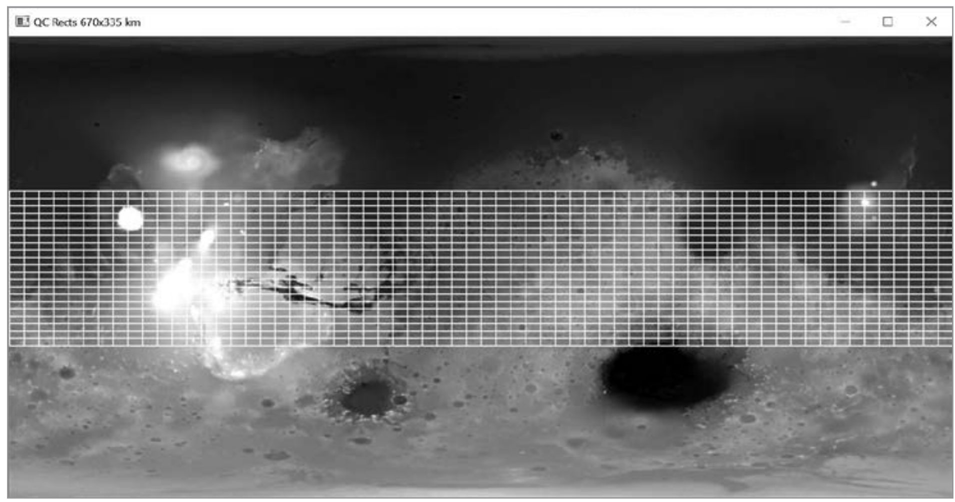


Рис. 7.10. Все 1260 прямоугольников, нарисованных методом draw_qc_rects()

Если сравнить рис. 7.10 с рис. 7.8, то можно заметить, что прямоугольники оказываются меньше, чем ожидалось. Причина в том, что мы передвигали их в направлении вдоль и вниз, используя половину ширины и высоты, чтобы они накладывались друг на друга.

Упорядочивание статистических данных и оценка прямоугольников

В листинге 7.6 при помощи класса `Search` определяется метод для поиска прямоугольников с лучшими потенциальными зонами для посадки. Этот метод упорядочивает словари, содержащие статистические данные для прямоугольников, создает списки лучших образцов на основе данных о высоте неровностей профиля и стандартного отклонения, после чего отбирает прямоугольники, попавшие в оба эти списка. Именно они — лучшие при отборе места для посадки, так как все эти прямоугольные области имеют наименьшие показатели высоты неровностей профиля и стандартного отклонения.

Листинг 7.6. Упорядочивание и отбор прямоугольников на основе их статистических данных

site_selector.py, часть 6

```
    def sort_stats(self):
        """Сортировка словарей по значениям и создание списков из N лучших
        ключей"""
        ptp_sorted = (sorted(self.rect_ptps.items(), key=lambda x: x[1]))
        self.ptp_filtered = [x[0] for x in ptp_sorted[:NUM_CANDIDATES]]
        std_sorted = (sorted(self.rect_stds.items(), key=lambda x: x[1]))
        self.std_filtered = [x[0] for x in std_sorted[:NUM_CANDIDATES]]
```

```
        for rect in self.std_filtered:
            if rect in self.ptp_filtered:
                self.high_graded_rects.append(rect)
```

Определяем метод `sort_stats()`. Упорядочиваем словарь `rect_stats` с помощью лямбда-функции, сортирующей уже значения, а не ключи. Эти значения в словаре представляют измерения высоты неровностей профиля. Таким образом, должен получиться список кортежей, где номер прямоугольника находится в индексе 0, а значение высоты неровностей профиля — в индексе 1.

Затем с помощью спискового включения заполняем атрибут `self.ptp.filtered` номерами прямоугольников в списке `ptp_sorted`. Выполняем нарезку по индексам, чтобы отобрать только 20 значений, согласно константе `NUM_CONSTANT`. Теперь у нас есть 20 прямоугольников с наименьшими показателями высоты неровностей профиля. Повторяем тот же базовый код для стандартного отклонения, создавая список из 20 прямоугольников с наименьшими показателями.

Завершаем метод перебором номеров прямоугольников в списке `std_filtered` и их сравнением с прямоугольниками из списка `ptp_filtered`. Совпадающие номера добавляем в атрибут экземпляра `high_graded_rects`, созданного ранее методом `_init_()`.

Рисование отобранных прямоугольников на карте

Листинг 7.7 по-прежнему продолжает класс `Search`, здесь определяется метод, рисующий 20 лучших прямоугольников на полутоновой карте MOLA. Этот метод мы вызываем из функции `main()`.

Листинг 7.7. Рисование отобранных прямоугольников и линий широты на карте MOLA

site_selector.py, часть 7

```
    def draw_filtered_rects(self, image, filtered_rect_list):
        """Нарисовать прямоугольники в списке на изображении и вернуть
        изображение"""
        img_copy = image.copy()
        for k in filtered_rect_list:
            cv.rectangle(img_copy,
                         (self.rect_coords[k][0], self.rect_coords[k][1]),
                         (self.rect_coords[k][2], self.rect_coords[k][3]),
                         (255, 0, 0), 1)
            cv.putText(img_copy, str(k),
                       (self.rect_coords[k][0] + 1, self.rect_coords[k][3]- 1),
                       cv.FONT_HERSHEY_PLAIN, 0.65, (255, 0, 0), 1)

      ❶ cv.putText(img_copy, '30 N', (10, LAT_30_N - 7),
```

```
                 cv.FONT_HERSHEY_PLAIN, 1, 255)
        cv.line(img_copy, (0, LAT_30_N), (IMG_WIDTH, LAT_30_N),
                (255, 0, 0), 1)
        cv.line(img_copy, (0, LAT_30_S), (IMG_WIDTH, LAT_30_S),
                (255, 0, 0), 1)
        cv.putText(img_copy, '30 S', (10, LAT_30_S + 16),
                   cv.FONT_HERSHEY_PLAIN, 1, 255)

        return img_copy
```

Начинаем с определения метода, который в данном случае получает несколько аргументов. Помимо `self` ему потребуется загруженное изображение и список номеров прямоугольников. Используем локальную переменную, чтобы скопировать изображение, после чего начинаем перебирать номера прямоугольников в `filtered_rect_list`. Каждый цикл рисует прямоугольник, обращаясь по его номеру к координатам углов в словаре `rect_coords`.

Для того чтобы различать прямоугольники, мы с помощью метода OpenCV `putText()` указываем номер каждого в его нижнем левом углу. Для этого метода надо указать изображение, текст (в виде строки), координаты верхнего левого x и нижнего правого x, шрифт, ширину строки и цвет.

Далее указываем границы широт, начиная с 30° северной широты ❶. Затем с помощью метода OpenCV `line()` рисуем линию. В качестве аргументов этот метод получает изображение, пару координат (x, y) для начала и конца линии, цвет и толщину. Повторяем эти базовые инструкции для 30° южной широты.

Завершаем метод возвращением подписанного изображения. Лучшие прямоугольные области, исходя из статистических данных о высоте неровностей профиля (PTP) и стандартного отклонения (STD), показаны на рис. 7.11 и 7.12 соответственно.

Это 20 наиболее удачных площадок, выбранных на основе каждой статистики. Это не значит, что они всегда совпадают. Площадка с наименьшим стандартным отклонением может не появиться на рисунке, построенном на основе данных о высоте неровностей профиля, из-за наличия одного небольшого кратера. Чтобы найти самые плоские, ровные площадки, нужно вычленить те, которые присутствуют на обоих рисунках.

Создание заключительного, цветного, изображения

В листинге 7.8 завершается класс `Search`. Он определяет метод для отбора оптимальных прямоугольных площадок. Здесь с помощью `tkinter` создается окно обобщения, где прямоугольники размещаются на цветной карте MOLA.

Также выводятся статистические данные для прямоугольников (rect) в виде текстовых объектов под изображением. Это добавляет работы, но зато такое решение выглядит аккуратнее, чем размещение обобщенных статистик прямо на изображении с помощью OpenCV.

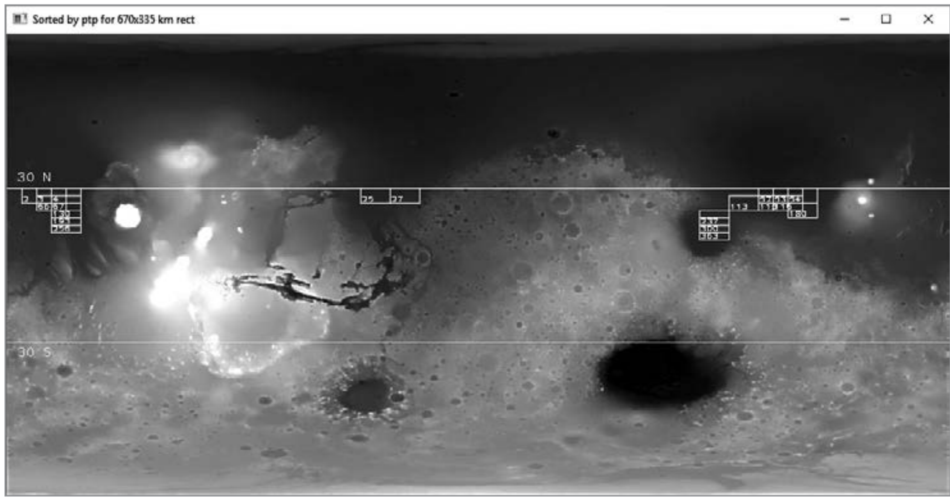


Рис. 7.11. 20 площадок с наименьшими показателями высоты неровностей профиля PTP

Рис. 7.12. 20 площадок с наименьшим стандартным отклонением STD

Листинг 7.8. Создание заключительного отображения с использованием цветной карты MOLA

site_selector.py, часть 8

```
    def make_final_display(self):
        """Используем Tk для показа карты итоговых прямоугольников и вывода
        их статистик"""
        screen.title('Sites by MOLA Gray STD & PTP {} Rect'.format(self.name))

        img_color_rects = self.draw_filtered_rects(IMG_COLOR,
                                                   self.high_graded_rects)

      ❶ img_converted = cv.cvtColor(img_color_rects, cv.COLOR_BGR2RGB)
        img_converted = ImageTk.PhotoImage(Image.fromarray(img_converted))
        canvas.create_image(0, 0, image=img_converted, anchor=tk.NW)

      ❷ txt_x = 5
        txt_y = IMG_HT + 20
        for k in self.high_graded_rects:
            canvas.create_text(txt_x, txt_y, anchor='w', font=None,
                               text="rect={} mean elev={:.1f}
                                     std={:.2f} ptp={}"
                           .    format(k, self.rect_means[k],
                                       self.rect_stds[k],
                                       self.rect_ptps[k]))
            txt_y += 15
          ❸ if txt_y >= int(canvas.cget('height')) - 10:
                txt_x += 300
                txt_y = IMG_HT + 20
        canvas.pack()
        screen.mainloop()
```

После определения метода даем окну `screen` название, связывающее его с именем объекта поиска: «Sites by MOLA Gray STD & PTP {} Rect» («Участки на полутоновой карте MOLA в виде прямоугольников, выбранных по значениям STD и PTP»).

Затем, чтобы вывести финальное изображение в цвете, создаем локальную переменную `img_color_rects` и вызываем метод `draw_filtered_rects()`. Ему передаем цветное изображение MOLA и список отобранных прямоугольников. Он вернет цветное изображение с отобранными прямоугольниками и границами широт.

Прежде чем разместить это новое изображение в `canvas`, нужно преобразовать цветовую схему из OpenCV варианта Blue-Green-Red (BGR) в используемую `tkinter` схему Red—Green—Blue (RGB). Делаем это с помощью метода OpenCV `cvtColor()`, которому передаем переменную изображения и флаг `COLOR_BGR2RGB` ❶. Результат называем `img_converted`.

На этом этапе изображение по-прежнему является массивом NumPy. Чтобы преобразовать tkinter-совместимое изображение, нужно использовать класс PhotoImage PIL модуля ImageTK и метод fromarray() модуля Image. Передаем этому методу переменную RGB-изображения, созданную на предыдущем шаге.

Подготовив изображение для tkinter, помещаем его в canvas, используя метод create_image(). Передаем ему координаты верхнего левого угла холста (0, 0), конвертированное изображение и якорь с точкой привязки NW (северо-запад).

Теперь осталось добавить текст описания. Присваиваем координаты для нижнего левого угла первого текстового объекта ❷. Затем перебираем номера прямоугольников в списке отобранных образцов. С помощью метода create_text() помещаем текст в canvas. Передаем этому методу пару координат, выровненную по левому краю точку привязки якоря, предустановленный шрифт и текстовую строку. Статистические данные получаем через обращение к разным словарям по номеру прямоугольника, обозначенному как k «ключ».

После того как отображен каждый текстовый объект, увеличиваем координату *y* текстового блока на 15. Затем пишем условную конструкцию, чтобы проверить, размещается ли текст в пределах 10 или более пикселей от нижнего края canvas ❸. Высоту canvas можно получить при помощи метода cget().

Если текст слишком близок к нижней границе canvas, нужно начинать новый столбец. Смещаем переменную txt_x на 300 и сбрасываем txt_y до высоты изображения плюс 20.

Завершаем метод упаковкой canvas и последующим вызовом цикла mainloop() объекта screen. Упаковывание оптимизирует размещение объектов на canvas. Бесконечный цикл mainloop() запускает tkinter, ожидает событие и обрабатывает его, пока не закроется окно.

ПРИМЕЧАНИЕ

Высота цветного изображения (506 пикселей) несколько больше, чем высота полутонового (501 пиксель). Я решил этот нюанс проигнорировать, но если вы склонны к педантизму, то можете с помощью OpenCV уменьшить высоту цветного образца, используя IMG_COLOR = cv.resize (IMG_COLOR, (1024, 501), interpolation=cv.INTER_AREA).

Выполнение программы через main()

В листинге 7.9 мы определяем функцию main() для выполнения программы.

Листинг 7.9. Определение и вызов функции main() для выполнения программы *site_selector.py*, часть 9

```
def main():
    app = Search('670x335 km')
    app.run_rect_stats()
    app.draw_qc_rects()
    app.sort_stats()
    ptp_img = app.draw_filtered_rects(IMG_GRAY, app.ptp_filtered)
    std_img = app.draw_filtered_rects(IMG_GRAY, app.std_filtered)

❶   cv.imshow('Sorted by ptp for {} rect'.format(app.name), ptp_img)
    cv.waitKey(3000)
    cv.imshow('Sorted by std {} rect'.format(app.name), std_img)
    cv.waitKey(3000)

    app.make_final_display() # Включает в себя вызов mainloop().

❷ if_name == '_main__':
    main()
```

Начинаем с инстанцирования объекта `app` из класса `Search`. Присваиваем ему имя `670x335 km`, документируя таким образом размер исследуемых прямоугольных областей. Далее по порядку вызываем методы `Search`. Получаем статистические данные для прямоугольников и рисуем прямоугольники для контроля качества. Упорядочиваем полученные данные от наименьших значений к наибольшим, после чего рисуем прямоугольники с лучшими показателями высоты неровностей профиля и стандартного отклонения. Показываем результат ❶ и завершаем функцию созданием заключительного обобщенного изображения.

Возвращаемся в глобальное пространство, где добавляем код, который запускает программу в качестве импортируемого модуля или в автономном режиме ❷.

На рис. 7.13 показано итоговое изображение. Оно включает отобранные прямоугольники и сводку статистик, упорядоченных на основе стандартного отклонения.

Результаты

После создания заключительного отображения первым делом нужно произвести проверку на правильность, а именно убедиться, что прямоугольники располагаются в допустимом диапазоне широт и вписываются в границы высоты, а также находятся на ровной местности. Аналогичным образом прямоугольники, отобранные по высоте неровностей профиля и стандартному

отклонению (рис. 7.11 и рис. 7.12 соответственно), должны вписываться в эти ограничения, и большинство из них должно занимать те же прямоугольные области.

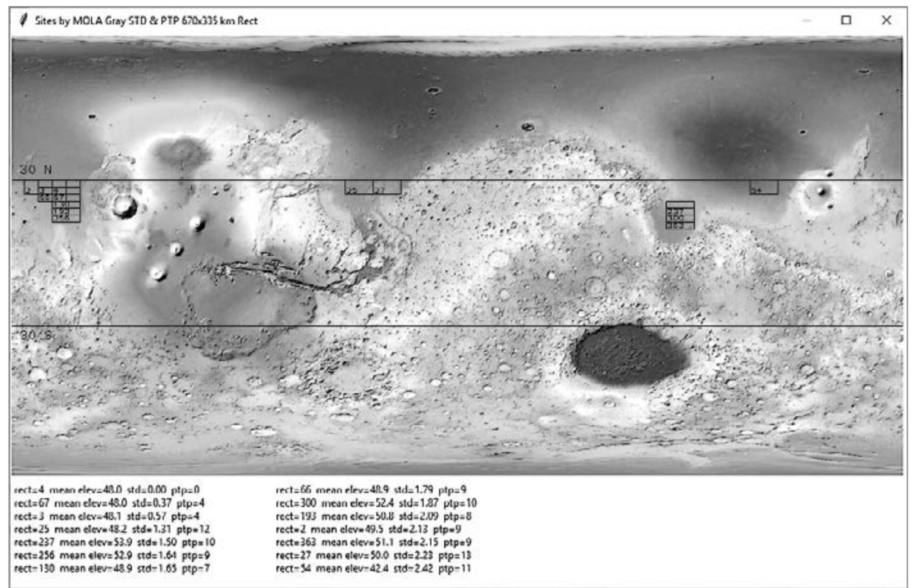


Рис. 7.13. Заключительное изображение содержит оптимальные прямоугольные площадки и сводные статистические данные, упорядоченные по стандартному отклонению std

Как говорилось ранее, площадки на рис. 7.11 и 7.12 совпадают не идеально. Причина — в использовании двух разных метрик для определения ровности. Тем не менее можно быть уверенными в том, что те из них, которые совпадают, будут наиболее ровными из всех.

На заключительном рисунке собраны все оптимальные прямоугольные области, и их концентрация у западного края карты особенно обнадеживает. Здесь наблюдается самая ровная поверхность во всей зоне поиска (рис. 7.14), и программа ясно это распознала.

Наш проект ориентирован на безопасность, но выбор мест посадки для большинства миссий определяется в том числе и их научными задачами. В практических проектах в конце главы у вас появится возможность добавить в уравнение отбора мест посадки дополнительный фактор — геологию.

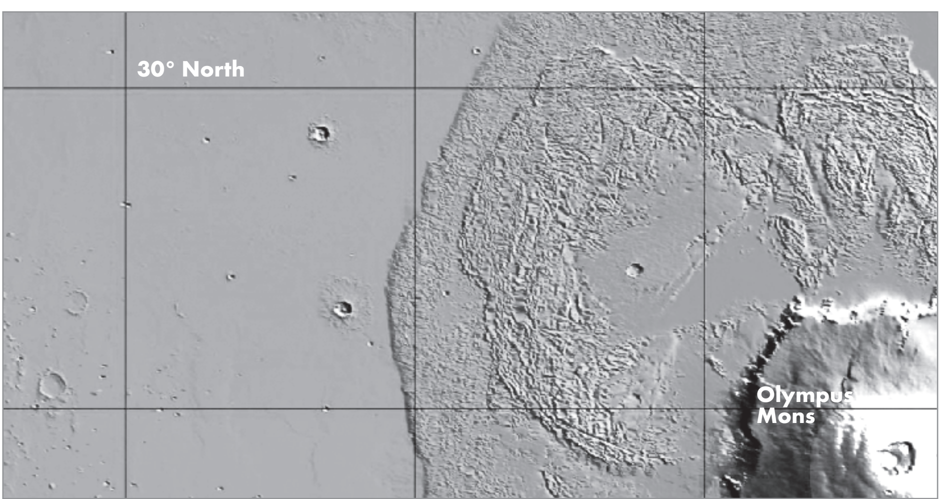


Рис. 7.14. Очень ровная поверхность на запад от полей лавы возле горы Олимп

Итоги

В этой главе мы использовали Python, OpenCV, Python Imaging Library, `NumPy` и `tkinter` для загрузки, анализа и показа изображений. Так как OpenCV рассматривает изображения в виде массивов `NumPy`, можно легко извлекать из частей изображения информацию и оценивать ее при помощи научных библиотек Python.

Использованный нами датасет обеспечил быстрое скачивание и выполнение. Интерн в реальной лаборатории, конечно, использовал бы более крупный и точный набор данных, состоящий из миллионов реальных измерений высоты, но нам нужно было лишь изучить сам процесс, затратив немного усилий и получив удовлетворительные результаты.

Дополнительная литература

У Jet Propulsion Laboratory есть несколько коротких и забавных видео о посадке на Марс. Онлайн вы их найдете по запросу «*Mars in a Minute: How Do You Choose a Landing Site?*», а также «*Mars in a Minute: How Do You Land on Mars?*».

Книга «Mapping Mars: Science, Imagination, and the Birth of a World» (Picador, 2002), написанная Оливером Мортоном (Oliver Morton), посвящена истории современного изучения Марса, включая создание карты MOLA.

«The Atlas of Mars: Mapping Its Geography and Geology» (Cambridge University Press, 2019) Кеннета Кольза (Kenneth Coles), Кеннета Танака (Kenneth Tanaka)

и Филипа Кристенсена (Philip Christensen) представляет выдающийся универсальный справочный атлас Марса, содержащий карты минералогии, топографии, геологии, термальных свойств, приповерхностного водяного льда и не только.

Страница данных карты MOLA, использованной в проекте 10, находится на https://astrogeology.usgs.gov/search/map/Mars/GlobalSurveyor/MOLA/Mars_MGS_MOLA_DEM_mosaic_global_463m/.

Подробные данные о Марсе доступны на сайте Mars Orbital Data Explorer, созданном PDS Geoscience Node в Университете Вашингтона в Сент-Луисе (https://ode.rsl.wustl.edu/mars/index.aspx).

Практический проект: убедимся, что рисунки становятся частью изображения

Напишите программу Python, которая проверяет, становятся ли добавленные на изображение рисунки, такие как текст, линии, прямоугольники и т. д., частью этого изображения. Используйте `NumPy` для вычисления среднего значения, а также статистик стандартного отклонения и высоты неровностей профиля для прямоугольной области на полутоновом изображении MOLA. Но при этом не рисуйте контуры прямоугольника. Затем проведите вокруг этой области белую линию и еще раз просчитайте статистику. Совпадают ли два полученных результата?

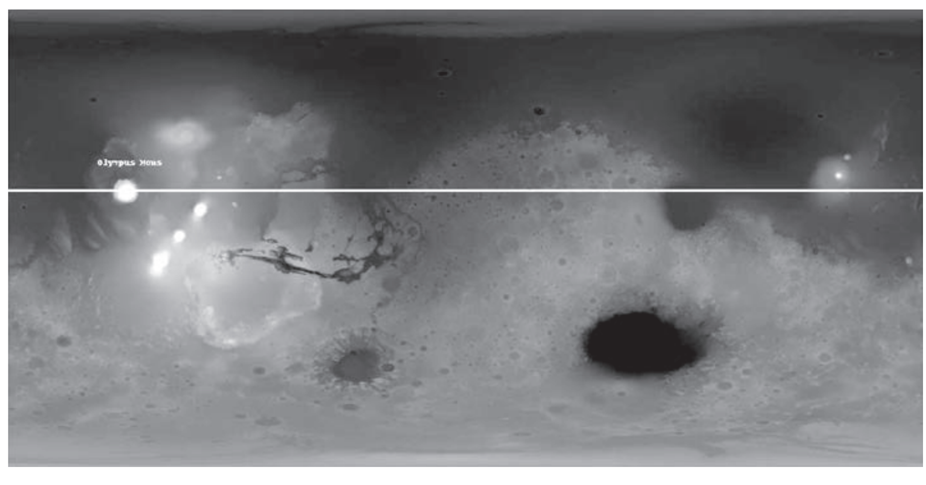
Решение под названием practice_confirm_drawing_part_of_image.py находится в приложении к книге или в каталоге Chapter_7, доступном для скачивания по адресу https://nostarch.com/real-world-python/.

Практический проект: визуализация профиля высот

Профиль высот — это двухмерное представление ландшафта в поперечном срезе. Это вид сбоку рельефа поверхности вдоль линии, нарисованной между некоторыми точками на карте. Геологи используют такие профили для изучения поверхности и визуализации ее топографии. Для данного практического проекта нарисуйте профиль с запада на восток вдоль линии, которая проходит через самый большой вулканический кратер в Солнечной системе, гору Олимп (рис. 7.15).

Используйте карту *Mars MGS MOLA — MEX HRSC Blended DEM Global 200m v2*, показанную на рис. 7.15. У этой версии более качественное поперечное разрешение, чем у использованной в проекте 10. В ней также используется полный диапазон

высот в данных MOLA. Ее копию под названием mola_1024x512_200mp.jpg можете найти в каталоге Chapter_7, доступном для скачивания с сайта книги. Решение же, practice_profile_olympus.py, находится в том же каталоге и в приложении к книге.



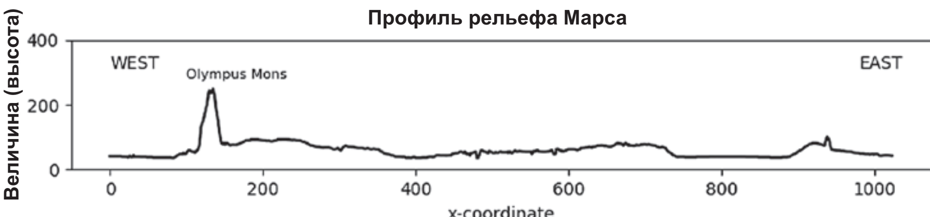


Рис. 7.15. Увеличенный по вертикали профиль через гору Олимп в направлении с запада на восток

Практический проект: отображение в 3D

Марс является несимметричной планетой: на южном полушарии доминируют испещренные кратерами высокогорья, а северное отличается ровными, плоскими долинами. Чтобы сделать это более наглядным, используйте предоставляемую `matplotlib` функциональность построения 3D-графиков для изображения mola_1024x512_200mp.jpg, которое использовали в предыдущем практическом проекте (рис. 7.16).

Библиотека `matplotlib` предоставляет точки, линии, контуры, каркасы модели и плоскости для создания 3D-графиков. Такие графики хотя и получаются

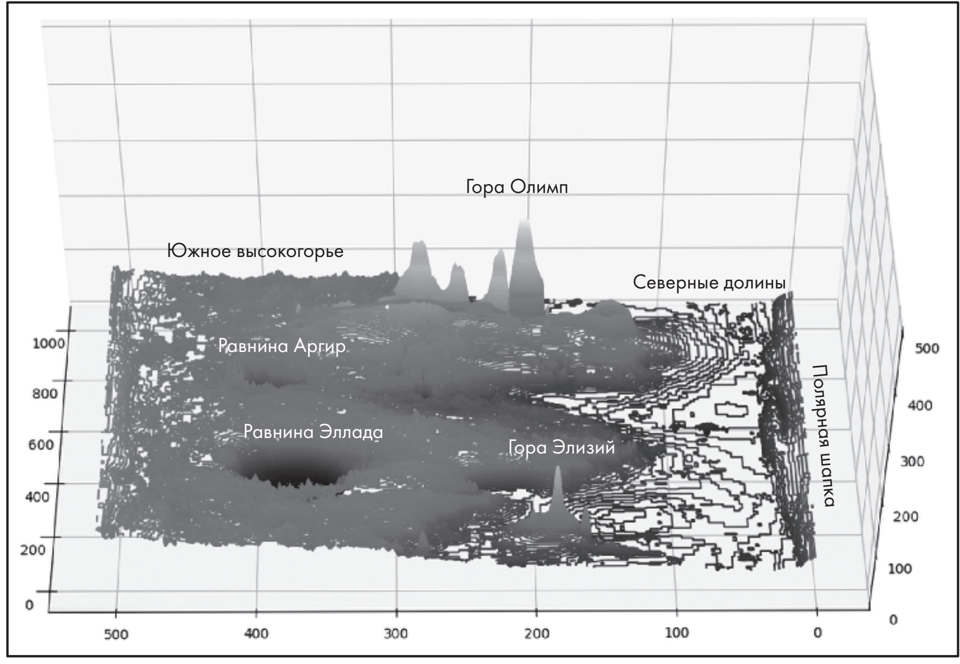


Рис. 7.16. 3D-график контуров Марса с ориентацией на запад

несколько грубоваты, зато генерируются быстро. Также можно использовать мышь для интерактивного захвата точек на графике и смены угла обзора. Подобные графики особенно полезны для тех, кому сложно визуализировать топографию на основе 2D-карт.

На рис. 7.16 более крупная вертикальная шкала делает наглядным изменение в высоте с юга на север. Здесь также несложно отметить самую высокую гору (Олимп) и самый глубокий кратер (равнина Эллада).

Можете воссоздать график с рис. 7.16 без аннотации с помощью программы practice_3d_plotting.py, которую найдете в приложении к книге или в каталоге Chapter_7, доступном для скачивания с сайта книги. Изображение карты вы найдете там же.

Практический проект: совмещение карт

Создайте новый проект, который придаст научность процессу выбора посадочных площадок. Совместите карту MOLA с цветной геологической картой

и найдите наиболее ровные прямоугольные области среди вулканических отложений в провинции Фарсида (показана стрелкой на рис. 7.17).

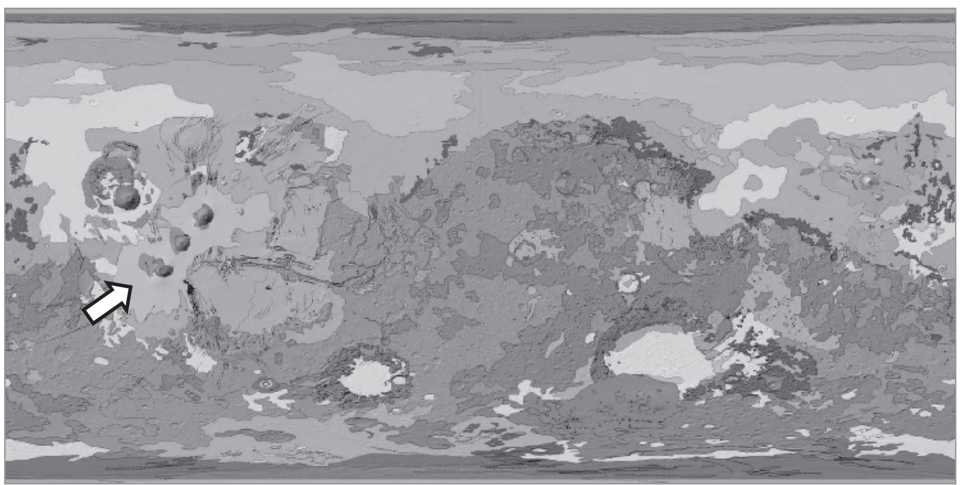


Рис. 7.17. Геологическая карта Марса. Стрелка указывает на район вулканических отложений в провинции Фарсида

Поскольку провинция Фарсида располагается на значительной высоте, то вместо определения низин сосредоточьтесь на поиске наиболее плоских и ровных областей среди вулканических отложений. Чтобы выделить именно вулканические отложения, рассмотрите вариант бинаризации полутоновой версии карты. *Бинаризация* — это техника сегментации, которая разделяет передний и задний планы.

С ее помощью можно конвертировать полутоновое изображение в двоичное, где пиксели над или между указанными пороговыми значениями устанавливаются как 1, а все остальные как 0. Такое бинарное изображение позволяет отфильтровать карту MOLA, как показано на рис. 7.18.

Геологическую карту Mars_Global_Geology_Mariner9_1024.jpg вы найдете в каталоге Chapter_7, доступном для скачивания с сайта книги. Вулканические отложения окрашены в светло-розовый цвет. В качестве карты высот используйте mola_1024x512_200mp.jpg из практического проекта «Визуализация профиля высот» на с. 225.

Решения practice_geo_map_step_1of2.py и practice_geo_map_step_2of2.py можно найти в том же каталоге или приложении к книге. Сначала запустите программу practice_geo_map_step_1of2.py, чтобы сгенерировать фильтр для шага 2.

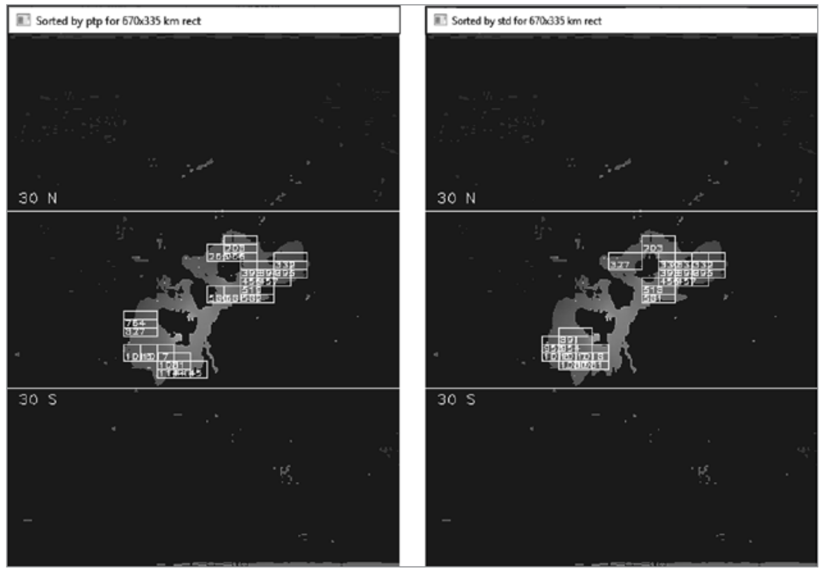


Рис. 7.18. Отфильтрованная карта MOLA над областью провинции Фарсида с прямоугольниками, полученными по высоте неровностей профиля ptp (*слева*) и стандартному отклонению std (*справа*)

Усложняем проект: три в одном

Доработайте проект «Извлечение профиля высот», чтобы профиль проходил через три вулкана, расположенных в провинции Фарсида, как показано на рис. 7.19.

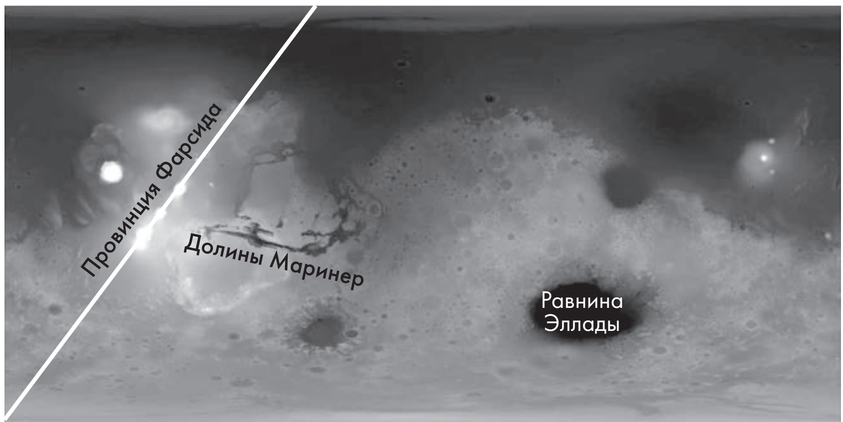


Рис. 7.19. Диагональный профиль, пролегающий через три вулкана в провинции Фарсида

Усложняем проект: перенос прямоугольников

Измените код site_selector.py, чтобы он учитывал размеры прямоугольников, которые не укладываются по ширине изображения MOLA. Один из способов — добавить код, который разделит прямоугольник на две части (одну вдоль правого края карты и вторую вдоль левого), вычислить для этих частей статистические данные и снова соединить их в цельный прямоугольник. Другой подход — дублирование изображения и «сшивание» его с оригинальным, как показано на рис. 7.20. В этом случае вам не придется разделять прямоугольники и нужно будет лишь решить, когда прекратить их перемещение по карте.

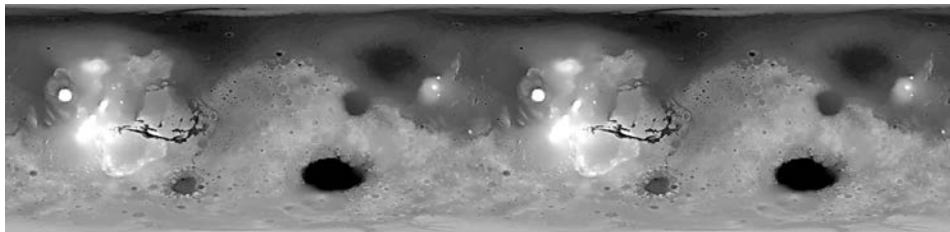


Рис. 7.20. Полутоновое изображение MOLA, дублированное и повторенное

Конечно же, вам не нужно дублировать всю карту. Понадобится только полоса вдоль восточного края, которая будет достаточно широка для охвата последнего накладывающегося прямоугольника.

8

Обнаружение далеких экзопланет

Экстрасолнечные планеты, коротко именуемые экзопланеты, представляют собой небесные тела, вращающиеся вокруг других солнц. К концу 2019 года было обнаружено более 4000 таких планет. В среднем получается по 150 единиц в год, начиная с первого подтвержденного открытия в 1992 году. Сегодня обнаружить удаленную планету едва ли сложнее, чем подхватить насморк, хотя человечеству понадобились десятки веков — вплоть до 1930 года, — чтобы открыть лишь восемь планет нашей Солнечной системы, а также Плутон.

Первую экзопланету астрономы обнаружили, наблюдая вызванные гравитацией колебания движения звезд. Применяемая же сегодня технология основана на едва заметном снижении яркости звезды ввиду прохождения на ее фоне экзопланеты. А с помощью мощных сверхсовременных устройств, наподобие космического телескопа Джеймса Уэбба, астрономы смогут непосредственно получать изображения экзопланет и изучать их вращение, смену сезонов года, наличие растительности, а также другие характеристики.

В этой главе мы с помощью OpenCV и `matplotlib` создадим симуляцию экзопланеты, проходящей перед ее солнцем. При этом мы запишем итоговую кривую изменения яркости (кривую блеска), после чего с ее помощью обнаружим планету и приблизительно оценим ее диаметр. Далее мы смоделируем возможный вид планеты через телескоп Уэбба. В практических проектах вы займетесь

исследованием нетипичных кривых блеска, которые могут представлять гигантские инопланетные мегаструктуры, построенные для аккумулирования энергии звезд.

Транзитная фотометрия

В астрономии *транзит* — это движение относительно малого небесного тела непосредственно между диском более крупного тела и наблюдателем. Когда это небольшое тело движется на фоне более крупного, последнее несколько теряет в яркости. Наиболее известные транзиты — это прохождение Меркурия и Венеры на фоне нашего Солнца (рис. 8.1).

Рис. 8.1. Облака и Венера (черная точка), проходящая перед Солнцем в июне 2012 года

С помощью современных технологий астрономы могут обнаружить даже малейшее уменьшение свечения удаленных звезд в процессе транзита на их фоне объектов. Такая техника, называемая *транзитной фотометрией*, позволяет построить график изменения яркости звезды в течение некоторого времени (рис. 8.2).

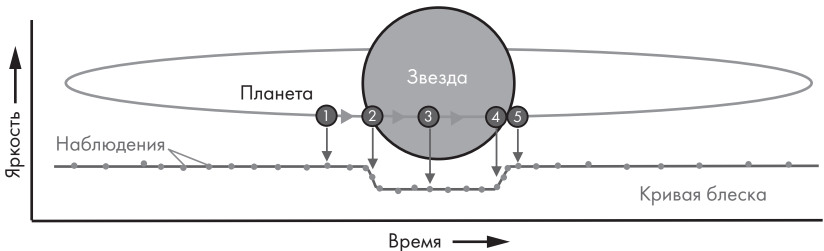


Рис. 8.2. Техника транзитной фотометрии для обнаружения экзопланет

На рис. 8.2 точки на графике кривой блеска показывают изменения яркости исходящего от звезды света. Когда планета еще не зашла в область диска звезды ❶, яркость максимальна. (Мы будем игнорировать свет, отражаемый экзопланетой по мере ее прохождения через эти фазы, что слегка увеличивает наблюдаемую яркость звезды.) По мере того как край диска планеты заходит на диск звезды ❷, ее яркость постепенно снижается, в результате чего на кривой блеска формируется нисходящий участок. Когда на фоне диска звезды оказывается вся планета ❸, кривая блеска представляет собой прямой участок, параллельный оси x. Такой ее вид сохраняется, пока планета не начнет выходить за противоположный край диска. В этот период времени формируется восходящий участок ❹, который заканчивается в момент полного выхода планеты за область диска ❺. Теперь кривая блеска снова превращается в прямую линию, показывающую максимальное значение яркости в любой момент времени.

Так как количество блокируемого в процессе транзита света пропорционально размеру диска планеты, можно вычислить ее радиус, используя следующую формулу:

$$R_p = R_s\sqrt{\text{Глубина}}.$$

Здесь R_p представляет радиус планеты, а R_s — радиус звезды. Астрономы определяют радиус звезды на основе расстояния до нее, ее яркости и цвета, который характеризует температуру. *Глубина* показывает полное изменение яркости во время транзита (рис. 8.3).

Конечно же, эти вычисления предполагают, что звезду заслоняла вся экзопланета, а не ее часть. Последнее может случиться, если экзопланета лишь частично захватывает верхний либо нижний край звезды (с нашего угла обзора). Мы рассмотрим этот случай в разделе «Эксперименты с транзитной фотометрией» на с. 242.

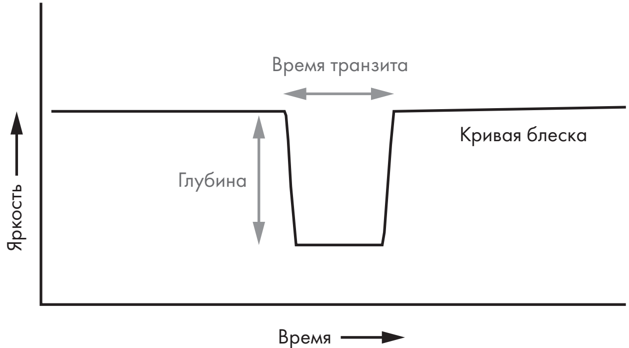


Рис. 8.3. Глубина представляет полное изменение яркости, наблюдаемое в кривой блеска

Проект #11. Симуляция транзита экзопланеты

Прежде чем полететь в Айдахо фотографировать Великое американское затмение в 2017 году, я подготовился. Было известно, что полная фаза, в течение которой Луна будет полностью закрывать Солнце, продлится всего 2 минуты 10 секунд. Так что времени на эксперименты, тесты и выяснение деталей на ходу не будет. Чтобы успешно заснять конус полутени, тень, протуберанцы и эффект бриллиантового кольца (рис. 8.4), мне нужно было знать, какое именно брать оборудование, какие настройки камеры использовать и когда эти явления должны произойти.

Схожим образом компьютерные симуляции помогают нам подготовиться к наблюдениям природных явлений. Они помогают понять, чего ожидать, когда ожидать и как откалибровать инструменты. В текущем проекте мы создадим симуляцию транзита экзопланеты. Эту симуляцию вы сможете выполнять с планетами различных размеров, чтобы понять влияние размера на кривую блеска. Позже с помощью данной симуляции мы оценим кривые блеска, относящиеся к полям астероидов и возможным инопланетным мегаструктурам.

ЗАДАЧА

Написать программу Python, которая будет симулировать транзит экзопланеты, отображать получающуюся кривую блеска и вычислять радиус этой планеты.

Рис. 8.4. Эффект бриллиантового кольца в конце фазы полного солнечного затмения в 2017 году

Стратегия

Для генерации кривой блеска потребуется возможность измерять изменения яркости. Обратимся к OpenCV и выполним математические операции над пикселями, такими как поиск среднего, минимального и максимального значений.

Вместо использования изображения реального транзита и звезды мы нарисуем круги на черном прямоугольнике аналогично тому, как рисовали прямоугольники на карте Марса в предыдущей главе. Для отражения графика кривой блеска можно использовать `matplotlib`, основную библиотеку Python для создания графиков. Ее мы установили в разделе «Установка `NumPy` и других научных библиотек с помощью pip» на с. 33 и уже использовали для создания графиков в главе 2.

Код для транзита

Программа transit.py с помощью OpenCV генерирует визуальную симуляцию экзопланеты, проходящей на фоне звезды, с помощью `matplotlib` создает

график итоговой кривой блеска и в конце оценивает размер планеты, используя уравнение планетарного радиуса со с. 233. Код можете ввести сами или скачать с https://nostarch.com/real-world-python/.

Импорт модулей и присваивание констант

В листинге 8.1 мы импортируем модули и присваиваем константы, представляющие вводимые пользователем данные.

Листинг 8.1. Импорт модулей и присваивание констант

transit.py, часть 1

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

IMG_HT = 400
IMG_WIDTH = 500
BLACK_IMG = np.zeros((IMG_HT, IMG_WIDTH, 1), dtype='uint8')
STAR_RADIUS = 165
EXO_RADIUS = 7
EXO_DX = 3
EXO_START_X = 40
EXO_START_Y = 230
NUM_FRAMES = 145
```

Импортируем модуль `math` для уравнения радиуса планеты, `NumPy` для вычисления яркости изображения, OpenCV для рисования симуляции и `matplotlib` для графического отображения кривой блеска. Затем присваиваем константы, которые представляют пользовательские значения.

Начинаем с высоты и ширины окна симуляции. Это окно будет черным прямоугольным изображением. Его создаем с помощью метода `np.zeros()`, который возвращает заполненный нулями массив установленной формы и типа.

Напомню, что изображения OpenCV — это массивы `NumPy`, элементы которых должны иметь одинаковый тип. Тип данных `uint8` представляет беззнаковое целое между 0 и 255. Полезный список других типов данных и их описания можете найти на странице https://numpy.org/devdocs/user/basics.types.html.

Далее присваиваем значения радиуса в пикселях для звезды и экзопланеты. OpenCV использует эти константы, когда рисует изображающие их круги.

Экзопланета будет двигаться перед звездой, поэтому нужно задать скорость ее движения. Константа `EXO_DX` инкрементирует позицию *x* планеты на три пикселя с каждым циклом программы, в результате чего планета будет смещаться слева направо.

Присваиваем две константы, устанавливая начальную позицию планеты. Затем присваиваем константу NUM_FRAMES для управления количеством обновлений симуляции. Несмотря на то что это число можно вычислить (IMG_WIDTH/EXO_DX), его присваивание позволяет точно подстроить продолжительность симуляции.

Определение функции main()

Код листинга 8.2 определяет функцию main() для выполнения программы. Можно определить main() в любом месте, но размещение ее в начале позволит ей предоставлять контекст для функций программы, определяемых далее. В составе main() можно также вычислить радиус экзопланеты, вложив соответствующее уравнение в вызов функции print().

Листинг 8.2. Определение функции main()

transit.py, часть 2

```
def main():
    intensity_samples = record_transit(EXO_START_X, EXO_START_Y)
    relative_brightness = calc_rel_brightness(intensity_samples)
    print('\nestimated exoplanet radius = {:.2f}\n'
          .format(STAR_RADIUS * math.sqrt(max(relative_brightness)
                                          - min(relative_brightness))))
    plot_light_curve(relative_brightness)
```

После определения main() создаем переменную intensity_samples и вызываем функцию record_transit(). *Интенсивность* (intensity) означает количество света, представленное численным значением пикселя. Функция record_transit() отрисовывает симуляцию на экране, измеряет ее интенсивность, добавляет полученные измерения в список intensity_samples, после чего этот список возвращает. Данной функции требуется начальная точка координат (x, y) экзопланеты. Передаем ей начальные константы EXO_START_X и EXO_START_Y, помещая тем самым планету в позицию, аналогичную ❶ на рис. 8.2. Обратите внимание, если существенно увеличить радиус экзопланеты, то может потребоваться сдвинуть начальную точку левее (отрицательные значения допускаются).

Далее создаем переменную relative_brightness и вызываем функцию calc_rel_brightness(). Эта функция вычисляет *относительную* яркость, которая равна измеренной интенсивности, поделенной на ее максимальное зарегистрированное значение. Она получает в качестве аргумента список измерений интенсивности, конвертирует их в относительную яркость и возвращает новый список.

Этот список значений относительной яркости мы используем для вычисления радиуса экзопланеты в пикселях с помощью уравнения со с. 233. Можно выполнить эти вычисления в рамках функции print(). Для сообщения ответа с точностью до двух десятичных знаков используем {:.2f}.

Завершается main() вызовом функции для рисования графика кривой блеска, которой передается список relative_brightness.

Регистрация транзита

Код листинга 8.3 определяет функцию для симуляции и регистрирования транзита. Он рисует звезду и экзопланету на изображении черного прямоугольника, после чего выполняет смещение экзопланеты. Здесь также происходит вычисление и отображение средней интенсивности изображения с каждым движением, добавление этого значения интенсивности в список и возвращение итогового списка в конце.

Листинг 8.3. Рисование симуляции, вычисление интенсивности изображения и возвращение ее значений в виде списка

transit.py, часть 3

```
def record_transit(exo_x, exo_y):
    """Нарисовать планету, проходящую мимо звезды, и вернуть список изменений
    интенсивности"""
    intensity_samples = []
    for _ in range(NUM_FRAMES):
        temp_img = BLACK_IMG.copy()
        cv.circle(temp_img, (int(IMG_WIDTH / 2), int(IMG_HT / 2)),
                  STAR_RADIUS, 255, -1)
      ❶ cv.circle(temp_img, (exo_x, exo_y), EXO_RADIUS, 0, -1)
        intensity = temp_img.mean()
        cv.putText(temp_img, 'Mean Intensity = {}'.format(intensity), (5, 390),
                   cv.FONT_HERSHEY_PLAIN, 1, 255)
        cv.imshow('Transit', temp_img)
        cv.waitKey(30)
      ❷ intensity_samples.append(intensity)
        exo_x += EXO_DX
    return intensity_samples
```

Функция record_transit() получает в качестве аргументов пару координат (x, y). Они показывают начальную позицию экзопланеты или, говоря точнее, пиксель, используемый в качестве центра для первого нарисованного круга. При этом он не должен пересекаться с кругом, представляющим звезду и размещенным по центру изображения.

Далее создаем пустой список для хранения измерений интенсивности. После этого начинаем цикл for, который использует константу NUM_FRAMES для повторения симуляции определенное число раз. Симуляция должна длиться немного дольше, чем требуется для полного выхода экзопланеты за пределы диска звезды. Таким образом, мы получим кривую блеска, включающую измерения после транзита.

Картинки и текст, которые мы размещаем на изображении при помощи OpenCV, становятся частью этого изображения. Следовательно, с каждым циклом нужно

заменять предыдущее изображение, копируя его оригинал BLACK_IMG в локальную переменную temp_img.

Теперь можно рисовать звезду, используя метод OpenCV circle(). Передаем ему временное изображение, координаты (x, y) для центра круга, соответствующего центру изображения, константу STAR_RADIUS, белый цвет для заполнения и толщину линии. Используя отрицательное значение для толщины, мы заполняем цветом весь круг.

Далее рисуем экзопланету. Берем координаты exo_x и exo_y в качестве начальной точки, константу EXO_RADIUS в качестве размера и черный цвет для заполнения ❶.

В этот момент нужно начинать регистрировать интенсивность изображения. Поскольку пиксели уже представляют интенсивность, нужно просто взять среднее по изображению. Количество проводимых измерений зависит от константы EXO_DX. Чем больше ее значение, тем быстрее движется экзопланета и тем реже будет зарегистрирована средняя интенсивность.

Для отображения вычисленной интенсивности на рисунке используем метод OpenCV putText(). Ему передаются временное изображение, текстовая строка, включающая сами измерения, координаты (x, y) для нижнего левого угла текстовой строки, шрифт, его размер и цвет.

Теперь называем окно Transit и отображаем его с помощью метода OpenCV imshow(). На рис. 8.5 показана итерация цикла.

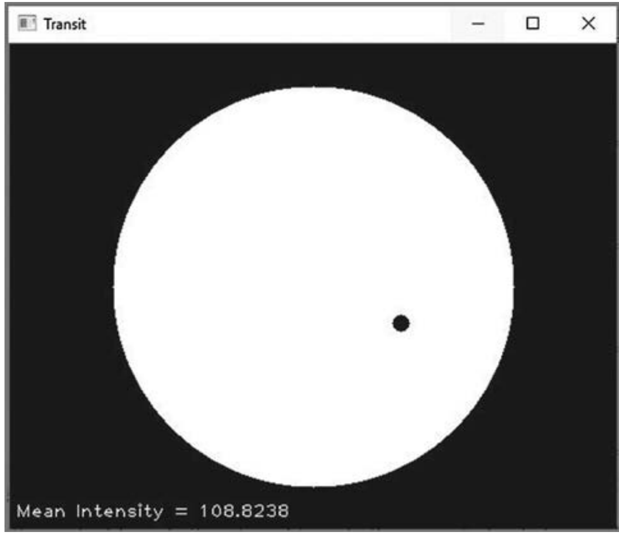


Рис. 8.5. Экзопланета, проходящая на фоне звезды

После вывода изображения используем метод OpenCV `waitKey()` для его обновления каждые 30 мс. Чем меньше переданное этому методу значение, тем быстрее экзопланета будет перемещаться.

Добавляем измеренное значение интенсивности в список `intensity_samples`, после чего продвигаем круг экзопланеты, увеличивая ее значение `exo_x` на константу `EXO_DX` ❷. Завершается функция возвращением списка значений средней интенсивности.

Вычисление относительной яркости и построение кривой блеска

Код листинга 8.4 определяет функцию для вычисления относительной яркости каждого значения интенсивности и построения графика кривой блеска. Затем следует вызов функции `main()`, если программа не используется в качестве модуля в другой программе.

Листинг 8.4. Вычисление относительной яркости, отображение графика кривой блеска и вызов main()

transit.py, часть 4

```
   def calc_rel_brightness(intensity_samples):
       """Вернуть список относительной яркости из списка значений
       интенсивности"""
       rel_brightness = []
       max_brightness = max(intensity_samples)
       for intensity in intensity_samples:
           rel_brightness.append(intensity / max_brightness)
       return rel_brightness

❶  def plot_light_curve(rel_brightness):
       """Построить график изменения относительной яркости в зависимости
       от времени"""
       plt.plot(rel_brightness, color='red', linestyle='dashed',
                linewidth=2, label='Relative Brightness')
       plt.legend(loc='upper center')
       plt.title('Relative Brightness vs. Time')
       plt.show()

❷  if __name__ == '__main__':
       main()
```

Кривые блеска отражают *относительную* яркость с течением времени так, что полностью незатемненная звезда имеет значение 1.0, а полностью затемненная — значение 0.0. Чтобы преобразовать измерения средней интенсивности в относительные значения, мы определяем функцию `calc_rel_brightness()`, которая получает в качестве аргумента список усредненных измерений интенсивности.

Внутри этой функции создаем пустой список для хранения конвертированных значений, после чего используем встроенную в Python функцию `max()` для поиска максимального значения в списке `intensity_samples()`. Для получения относительной яркости перебираем элементы этого списка и делим их на максимальное значение. По ходу дела добавляем результаты в список `rel_brightness`. Завершаем функцию возвращением нового списка.

Определяем вторую функцию для нанесения графика кривой блеска и передаем ей список `rel_brightness` ❶. Используем метод `plot()` библиотеки `matplotlib()`, передавая ему этот список, цвет линии, стиль линии, ее толщину и метку для легенды графика. Добавляем легенду и название графика, после чего его отображаем. Результат — на рис. 8.6.

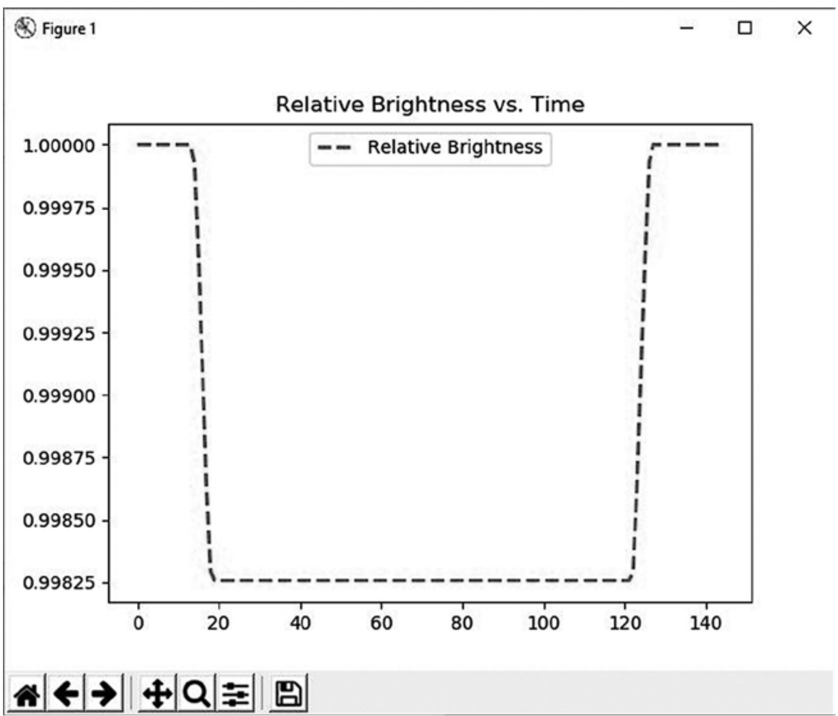


Рис. 8.6. Пример графика кривой блеска из transit.py

На первый взгляд отклонение яркости на этом графике может показаться чрезвычайно большим, но если поближе рассмотреть ось y, то можно видеть, что экзопланета уменьшила яркость звезды всего на 0.175 процента. Чтобы понять,

как это выглядит на графике абсолютной яркости звезды (рис. 8.7), добавим следующую строку непосредственно перед `plt.show()`:

```
plt.ylim(0, 1.2)
```

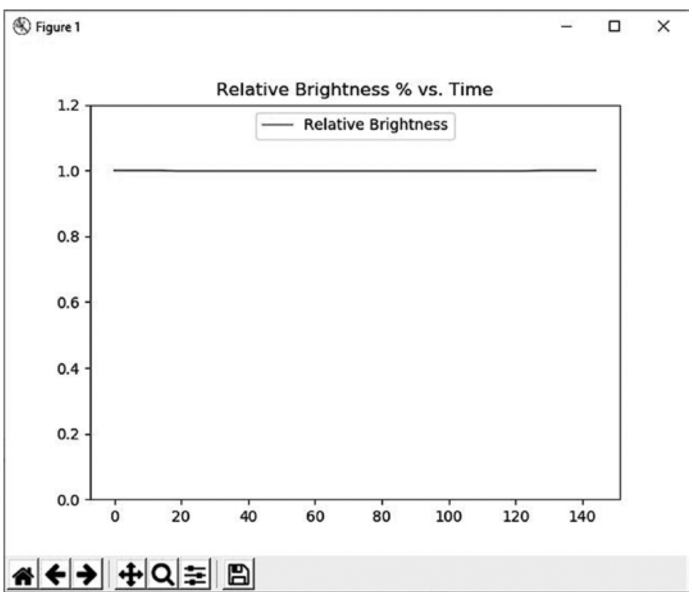


Рис. 8.7. Кривая блеска с рис. 8.6 с измененным масштабом оси y

Отклонение в кривой блеска, вызванное транзитом, едва заметно, но все-таки видно. Но мы все же не хотим ослепнуть, высматривая изменение этой кривой блеска, поэтому позволим `matplotlib` автоматически подстроить масштаб оси y, как на рис. 8.6.

Завершаем программу вызовом функции `main()` ❷. Помимо кривой блеска мы должны увидеть в оболочке вычисленный радиус экзопланеты.

```
estimated exoplanet radius = 6.89
```

Вот и все. Нам потребовалось менее 50 строк кода Python, чтобы создать средство для обнаружения экзопланет.

Эксперименты с транзитной фотометрией

Теперь, когда у вас есть рабочая симуляция, ее можно использовать для моделирования поведения транзитов, что позволит улучшить анализ реальных показателей в будущем. Одно из решений — выполнить множество возможных

кейсов и создать «атлас» ожидаемых ответов экзопланет. Исследователи смогут затем использовать этот атлас для интерпретации фактических кривых блеска.

Что, если, к примеру, плоскость орбиты экзопланеты наклонена относительно Земли, в результате чего она только частично проходит на фоне звезды во время транзита? Смогут ли исследователи обнаружить ее позицию на основе сигнатуры кривой блеска или она будет выглядеть как экзопланета меньшего размера, совершающая полноценный транзит?

Если вы выполните симуляцию с экзопланетой радиусом 7 и позволите ей захватить нижний край звезды, то должна получиться U-образная кривая блеска (рис. 8.8).

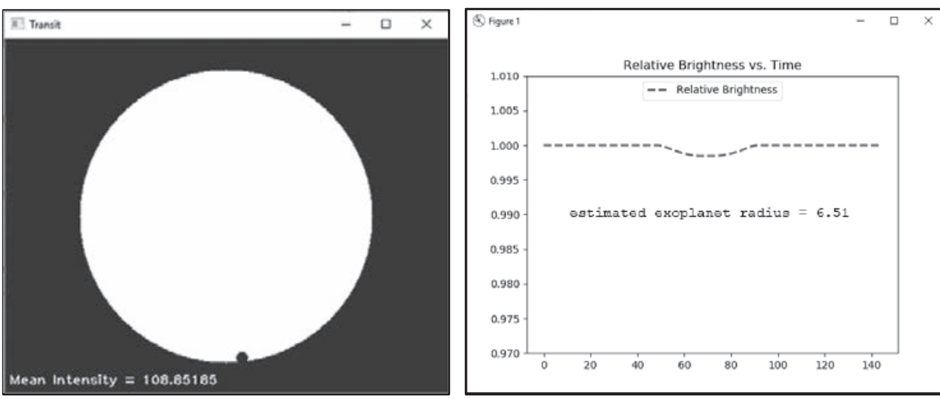


Рис. 8.8. Кривая блеска для экзопланеты с радиусом 7, которая проходит на фоне звезды только частично

Если еще раз выполнить симуляцию, но уже с экзопланетой радиусом 5 и позволить ей полностью пересечь диск звезды, то получится график, как на рис. 8.9.

Когда экзопланета проходит лишь по краю звезды, не пересекая ее диск полностью, область их наложения изменяется постоянно, генерируя U-образную кривую, как на рис. 8.8. Если же планета пересекает диск звезды, то основание кривой получается более плоским, как на рис. 8.9. А так как при частичном транзите на фоне звезды мы не видим весь диск планеты, то и способа измерить ее истинный размер у нас нет. Таким образом, если у кривой блеска неплоское дно, то и к оценке размеров планеты стоит относиться с недоверием.

Если вы выполните симуляцию с разными размерами экзопланет, то увидите, что по мере увеличения размера планеты кривая блеска становится глубже, у нее формируются более длинные стороны, так как яркость звезды сокращается более значительно (рис. 8.10 и 8.11).

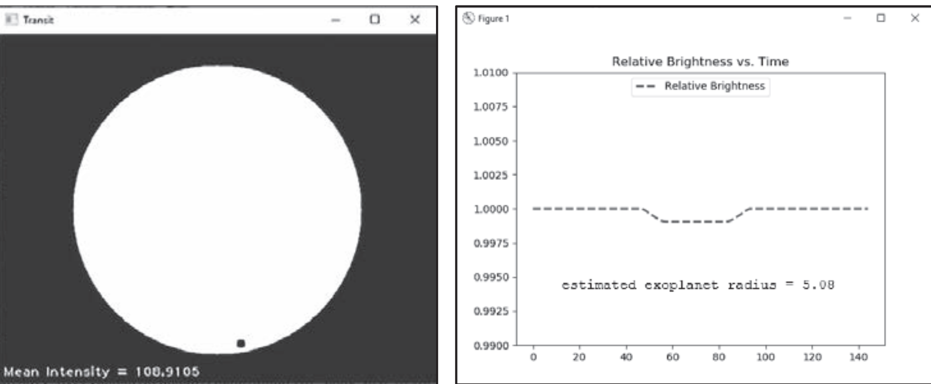


Рис. 8.9. Кривая блеска экзопланеты с радиусом 5, которая полностью пересекает диск своей звезды

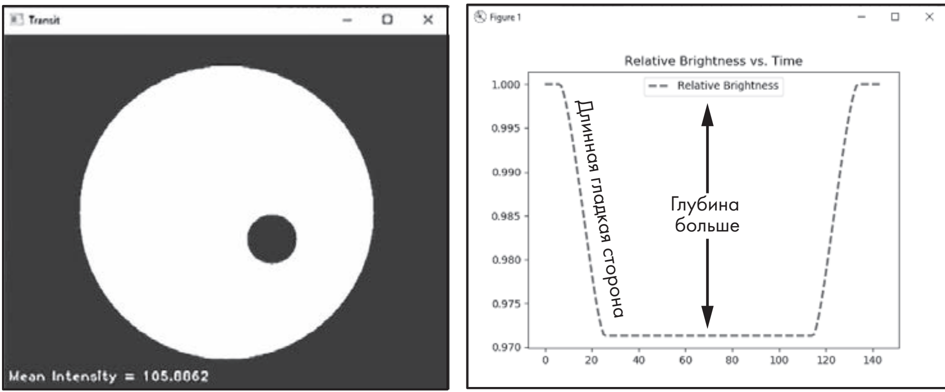


Рис. 8.10. Кривая блеска для EXO_RADIUS = 28

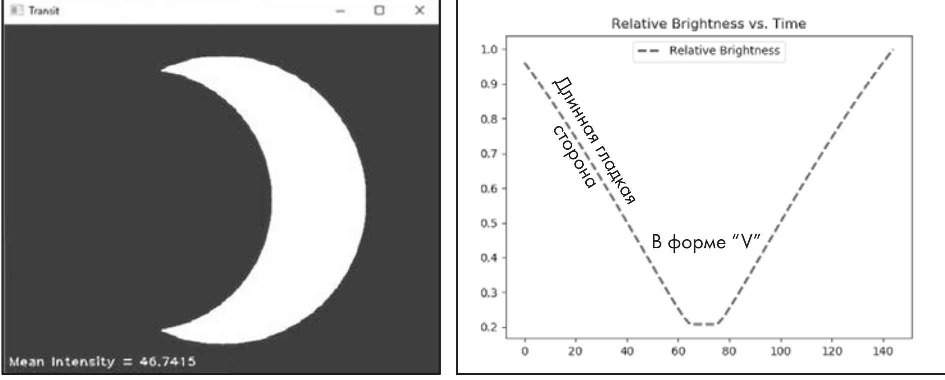


Рис. 8.11. Кривая блеска для EXO_RADIUS = 145

Поскольку экзопланеты — круглые, с гладкими контурами, то их кривые блеска имеют гладкие стороны, значения на которых постепенно увеличиваются или уменьшаются. Это важный факт, так как при поиске экзопланет астрономы неоднократно фиксировали неровные кривые. В разделе практических проектов в конце главы вы используете созданную программу для изучения кривых блеска необычной формы, которая может объясняться внеземными инженерными конструкциями.

Проект #12. Получение изображений экзопланет

К 2025 году три мощных телескопа — два на Земле и один в космосе — при помощи инфракрасного и видимого спектра будут искать экзопланеты размером с Землю. В лучшем сценарии такая экзопланета будет выглядеть как единичный насыщенный пиксель, который также влияет на интенсивность в соседних пикселях, но и этого достаточно для того, чтобы понять, вращается ли эта планета, имеет ли на поверхности континенты и моря, есть ли на ней смена сезонов и погодные явления, а также способна ли там существовать жизнь в известной нам форме.

В этом проекте мы будем симулировать анализ изображения, полученного с телескопов. В качестве заместителя отдаленной экзопланеты мы используем Землю. Таким образом, нам удастся легко соотнести проявление известных характеристик, таких как наличие континентов и океанов, с тем, что мы обнаружим в одном пикселе. Мы рассмотрим цветовую композицию и интенсивность отраженного света, на основе чего сделаем выводы относительно атмосферы, особенностей поверхности и вращения экзопланеты.

ЗАДАЧА

Написать программу на Python, которая пикселизует изображение Земли и рисует график интенсивности красного, зеленого и синего цветовых каналов.

Стратегия

Чтобы продемонстрировать возможности выделения особенностей рельефа и облачных формаций с помощью одного насыщенного пикселя, нам понадобятся всего два изображения: по одному для западного и для восточного полушарий. Специалисты NASA уже сфотографировали оба этих полушария из космоса (рис. 8.12) — что нам весьма на руку.

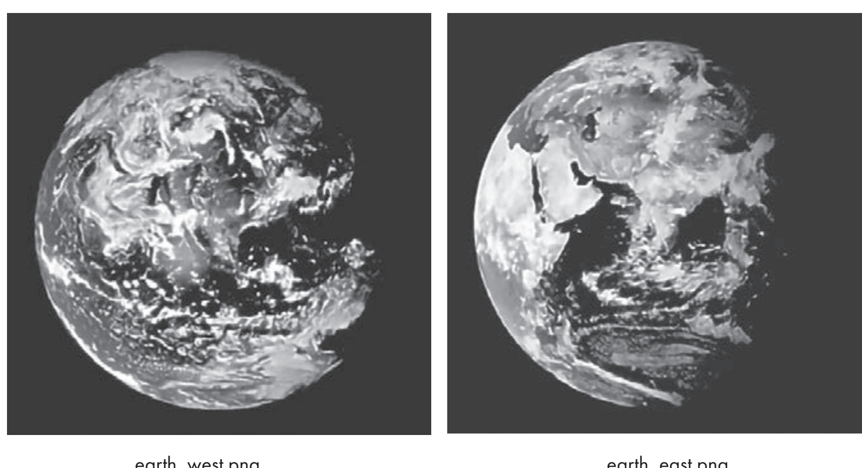


Рис. 8.12. Изображения западного и восточного полушарий Земли

Размер этих изображений составляет 474 × 474 пикселя, но такое разрешение слишком велико для будущих изображений экзопланет, где экзопланета будет располагаться всего на девяти пикселях, при этом лишь центральный пиксель будет полностью занят планетой (рис. 8.13).

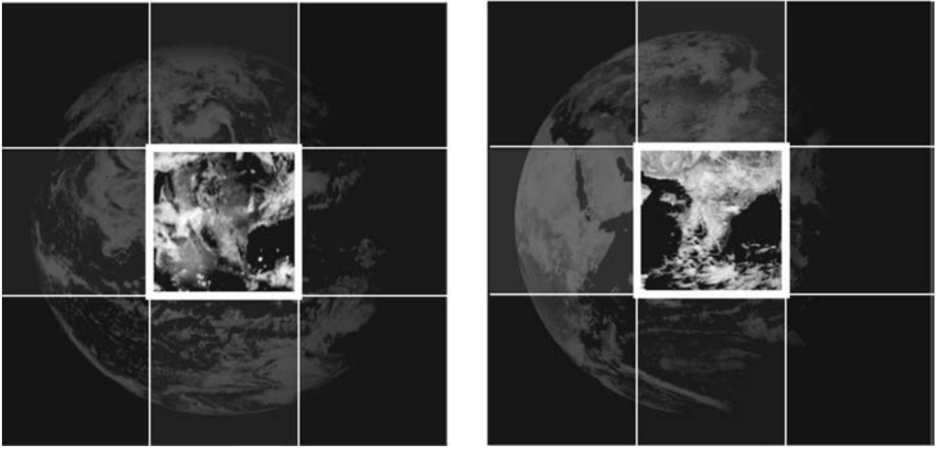


Рис. 8.13. Изображения earth_west.png и earth_east.png с наложением 9-пиксельной сетки

Нужно уменьшить эти изображения, отобразив их в массив 3 × 3. Поскольку OpenCV использует `NumPy`, сложности это не составит. Для обнаружения

изменений в рельефе экзопланеты нужно извлечь преобладающие цвета (синий, зеленый и красный). OpenCV позволит нам усреднить эти цветовые каналы, после чего мы отобразим результаты с помощью `matplotlib`.

Код для пикселизатора

Программа pixelator.py загружает два изображения Земли, изменяет их до размера 3 × 3 пикселя, после чего снова увеличивает изображения, но уже до 300 × 300 пикселей. Эти итоговые изображения нужны только для визуализации и содержат ту же цветовую информацию, что и фрагмент 3 × 3 пикселя. Затем программа усредняет цветовые каналы в обоих измененных изображениях и рисует график в виде удобных для сопоставления круговых диаграмм. Код программы и два сопутствующих изображения (earth_west.png и earth_east.png) можно скачать с сайта книги. Сохраните их в одном каталоге и не переименовывайте изображения.

Импорт модулей и уменьшение разрешения изображений

Код листинга 8.5 импортирует модули для рисования графика и обработки изображений, после чего загружает эти изображения и уменьшает их разрешение. Сначала он уменьшает каждое изображение до 9 пикселей в виде массива 3 × 3, а затем увеличивает эти упрощенные версии до размера 300 × 300 пикселей, делая их достаточно большими для визуального восприятия, и выводит их на экран.

Листинг 8.5. Импорт модулей, а также загрузка, уменьшение и последующий показ изображений

pixelator.py, часть 1

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

files = ['earth_west.png', 'earth_east.png']

for file in files:
    img_ini = cv.imread(file)
    pixelated = cv.resize(img_ini, (3, 3), interpolation=cv.INTER_AREA)
    img = cv.resize(pixelated, (300, 300), interpolation=cv.INTER_AREA)
    cv.imshow('Pixelated {}'.format(file), img)
    cv.waitKey(2000)
```

Импортируем `NumPy` и OpenCV для работы с изображениями и используем `matplotlib` для представления цветов в виде круговых диаграмм. Затем создаем список имен файлов, содержащих два изображения Земли.

Теперь перебираем файлы в созданном списке и с помощью OpenCV загружаем их в качестве массивов `NumPy`. Напомню, что OpenCV по умолчанию загружает цветные изображения, поэтому добавлять аргументы для этого не нужно.

Наша цель — уменьшить изображение Земли до одного насыщенного пикселя, окруженного пикселями с частичным насыщением. Для уменьшения изображения 474 × 474 пикселя до размера 3 × 3 пикселя мы используем метод OpenCV `resize()`. Сначала даем будущему новому изображению имя `pixelated` и передаем методу текущее изображение, новую ширину и высоту в пикселях, а также метод интерполяции. *Интерполяция* происходит, когда мы изменяем размер и используем известные данные для оценки значений в неизвестных точках. В документации OpenCV рекомендуется использовать для уменьшения изображений метод интерполяции `INTER_AREA` (подробнее — на странице https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html).

Итак, у нас получилось изображение, слишком мелкое для визуализации, поэтому мы снова изменяем его размер, но на этот раз до 300 × 300 пикселей, чтобы проверить результаты. Используем для этого методы интерполяции `INTER_NEAREST` или `INTER_AREA`, поскольку они сохраняют границы пикселей.

Отображаем изображение (рис. 8.14) и с помощью `waitKey()` приостанавливаем программу на две секунды.

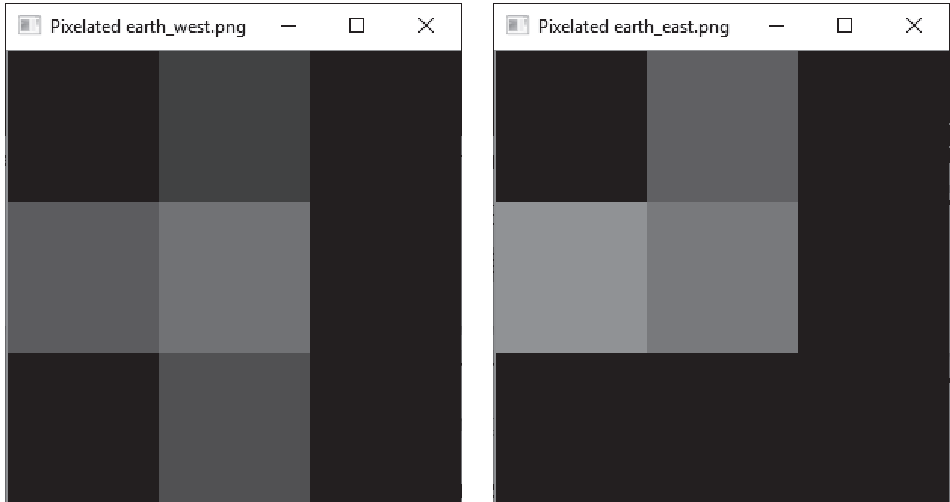


Рис. 8.14. Черно-белое представление пикселизированных цветных изображений

Имейте в виду, что нельзя восстановить изображения в их исходное состояние обратным изменением размера до 474 × 474. После усреднения значений пикселей до матрицы 3 × 3 все детали утрачиваются навсегда.

Усреднение цветовых каналов и создание круговых диаграмм

Продолжая цикл `for`, листинг 8.6 создает и отображает круговые диаграммы для синего, зеленого и красного цветов каждого пикселизированного изображения. Можете сравнить их, чтобы сделать выводы относительно погодных условий планеты, объема суши, ее вращения и т. п.

Листинг 8.6. Разделение и усреднение цветовых каналов, а также создание круговых диаграмм цветов

pixelator.py, часть 2

```
    b, g, r = cv.split(pixelated)
    color_aves = []
    for array in (b, g, r):
        color_aves.append(np.average(array))

    labels = 'Blue', 'Green', 'Red'
    colors = ['blue', 'green', 'red']
    fig, ax = plt.subplots(figsize=(3.5, 3.3)) # Размер в дюймах
❶   _, _, autotexts = ax.pie(color_aves,
                             labels=labels,
                             autopct='%1.1f%%',
                             colors=colors)
    for autotext in autotexts:
        autotext.set_color('white')
    plt.title('{}\n'.format(file))

plt.show()
```

С помощью метода OpenCV `split()` разделяем синий, зеленый и красный цветовые каналы пикселизированного изображения и записываем результаты в переменные `b`, `g`, `r`. Эти переменные являются массивами, и если вызвать `print(b)`, то вывод должен получиться таким:

```
[[ 49  93  22]
 [124 108  65]
 [ 52 118  41]]
```

Каждое число представляет пиксель — в данном случае значение синего цвета пикселя на изображении 3 × 3. Чтобы усреднить эти массивы, сначала создаем

пустой список, после чего перебираем массивы и вызываем предоставляемый `NumPy` метод усреднения, добавляя в созданный список его результаты.

Теперь можно создать из средних значений цветов каждого пикселизированного изображения круговые диаграммы. Присваиваем имена цветов переменной `labels`, которую используем для подписи секторов диаграммы. Далее указываем цвета, которые хотим задать в круговой диаграмме. Таким образом, мы переопределим предустановленные в `matplotlib` варианты. При создании диаграммы используем соглашение именования `fig, ax` для обозначения фигуры и оси, вызываем метод `subplots()` и передаем ему размер фигуры в дюймах.

Поскольку цвета изображений будут отличаться мало, нам нужно указать в каждой секции цвета его процент, чтобы сделать наглядной разницу между ними. К сожалению, по умолчанию `matplotlib` использует черный текст, который сложно различить на фоне темного фона. Чтобы это исправить, вызываем метод `ax.pie()` для создания круговых диаграмм и используем его список `autotexts` ❶. Этот метод возвращает три списка `autotexts`: один для секций диаграммы, второй для их меток и третий для числовых меток. Нам нужен только последний, поэтому первые два мы считаем неиспользуемыми переменными и обозначаем их нижним подчеркиванием.

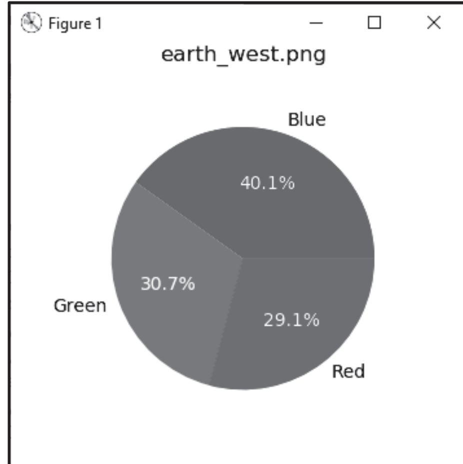
Передаем `ax.pie()` список средних значений цветов и список меток, а также устанавливаем его параметр `autopct` так, чтобы отображать числа с одним десятичным знаком. Если для этого параметра задать `None`, то список `autotexts` возвращен не будет. Завершаем аргументы передачей списка цветов для секций диаграммы.

Список `autotexts` для первого изображения выглядит так:

```
[Text(0.1832684031431146, 0.5713253822554821, '40.1%'),
Text(-0.5646237442340427,
-0.20297789891298565, '30.7%'), Text(0.36574010704848686,
-0.47564080364930983, '29.1%')
```

Каждый объект `Text` содержит в виде строки координаты (x, y) и значение в процентах. Все эти данные по-прежнему отображаются черным цветом, поэтому нужно перебрать эти объекты и изменить цвет на `white`, используя метод `set_color()`. Теперь осталось установить для диаграмм заголовки в соответствии с именами файлов и отобразить их (рис. 8.15).

Несмотря на похожесть этих диаграмм, они демонстрируют значительные различия. Если вы сравните оригинальные цветные изображения, то увидите, что earth_west.png включает больше океанов и должно давать большую долю синего цвета.



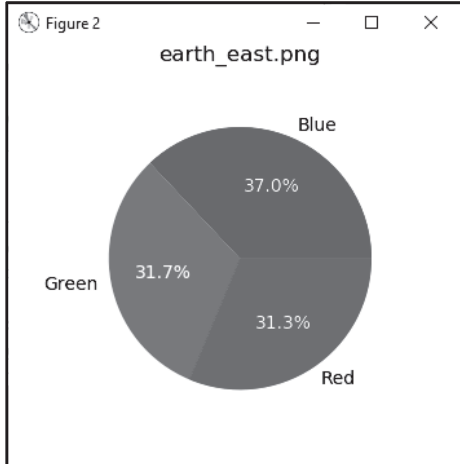


Рис. 8.15. Круговые диаграммы, созданные pixelator.py

Рисование графика для одного пикселя

Диаграммы на рис. 8.15 относятся ко всему изображению, то есть сюда входит и часть черного пространства. Для получения же чистого образца можно использовать один насыщенный пиксель в центре каждого изображения (листинг 8.7).

Этот код представляет отредактированную копию pixelator.py, в которой были подписаны измененные строки. Его цифровую копию вы найдете в каталоге Chapter_8 под названием pixelator_saturated_only.py.

Листинг 8.7. Рисование диаграмм для цветов центрального пикселя пикселизированного изображения

pixelator_saturated_only.py

```
import cv2 as cv
from matplotlib import pyplot as plt

files = ['earth_west.png', 'earth_east.png']

# Уменьшение изображения до размера 3x3 пикселя
for file in files:
    img_ini = cv.imread(file)
    pixelated = cv.resize(img_ini, (3, 3), interpolation=cv.INTER_AREA)
    img = cv.resize(pixelated, (300, 300), interpolation=cv.INTER_NEAREST)
    cv.imshow('Pixelated {}'.format(file), img)
    cv.waitKey(2000)

❶ color_values = pixelated[1, 1] # Выбор центрального пикселя
```

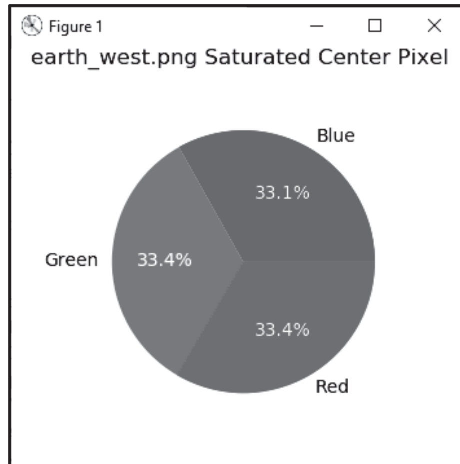
```
    # Создание диаграмм
    labels = 'Blue', 'Green', 'Red'
    colors = ['blue', 'green', 'red']
    fig, ax = plt.subplots(figsize=(3.5, 3.3)) # Размер в дюймах
  ❷ _, _, autotexts = ax.pie(color_values,
                              labels=labels,
                              autopct='%1.1f%%',
                              colors=colors)
    for autotext in autotexts:
        autotext.set_color('white')
  ❸ plt.title('{} Saturated Center Pixel \n'.format(file))

plt.show()
```

Четыре строки кода в листинге 8.6, которые разделяли изображение и усредняли цветовые каналы, можно заменить на одну строку ❶. Переменная `pixelated` — это массив `NumPy`, а `[1, 1]` представляет в этом массиве ряд 1, столбец 1. Напомню, что Python начинает отсчет от 0, значит, эти значения соответствуют центру массива 3 × 3. Если вывести переменную `color_values`, то мы получим другой массив.

```
[108 109 109]
```

Это значения синего, зеленого и красного цветовых каналов для центрального пикселя, и можно передать их непосредственно в `matplotlib` ❷. Изменим название диаграммы, чтобы показать, что анализируем только центральный пиксель ❸. На рис. 8.16 приведены итоговые диаграммы.



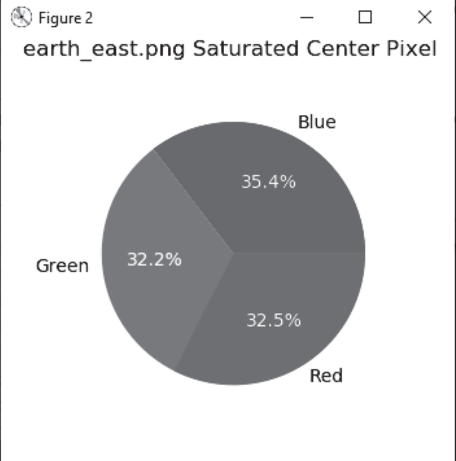


Рис. 8.16. Диаграммы, построенные на основе одного пикселя с помощью программы pixelator_saturated_only.py

Цвета западного и восточного полушарий на рис. 8.15 и 8.16 различаются незначительно, но мы знаем, что они реальны, потому что получили ответ с помощью *опережающего моделирования*. То есть мы сформировали результат из фактических наблюдений, поэтому уверены, что он осмыслен, повторяем и уникален.

При анализе реальной экзопланеты понадобится сделать как можно больше снимков. И если сходство и цветовые паттерны на снимках, сделанных в разное время, будут сохраняться, то можно исключить наличие таких непостоянных эффектов, как смена погодных условий. Если же цветовые паттерны будут изменяться прогнозируемым образом на протяжении значительных интервалов времени, то есть вероятность, что это проявление смены времен года, например наличие снега зимой и зеленая растительность в весенний и летний периоды.

Если подобные измерения повторять через относительно короткие отрезки времени, то можно сделать выводы о вращении планеты вокруг собственной оси. В практических проектах в конце главы у вас будет возможность вычислить протяженность дня на экзопланете.

Итоги

В этой главе мы использовали OpenCV, `NumPy` и `matplotlib` для создания изображений и анализа их свойств. Мы изменяли размер изображений, а также составляли графики, отображающие их интенсивность и цветовые каналы. С помощью коротких и простых программ Python мы симулировали важные методы, которые астрономы используют для обнаружения и изучения далеких экзопланет.

Дополнительная литература

«How to Search for Exoplanets», изданная Планетарным обществом (https://www.planetary.org/), дает хороший обзор техник, используемых для поиска экзопланет, с описанием сильных и слабых сторон каждой техники.

Руководство «Transit Light Curve Tutorial», написанное Эндрю Вандербургом (Andrew Vanderburg), знакомит с основами транзитной фотометрии и предоставляет ссылки на данные о транзитах, собранные обсерваторией Кеплера. Найти его можно по адресу https://www.cfa.harvard.edu/~avanderb/tutorial/tutorial.html.

«NASA Wants to Photograph the Surface of an Exoplanet» (Wired, 2020) Даниэла Оберхауза (Daniel Oberhaus) описывает усилия, необходимые, чтобы превратить Солнце в гигантскую линзу камеры для изучения экзопланет.

Книга «Dyson Spheres: How Advanced Alien Civilizations Would Conquer the Galaxy» (Space.com, 2014), написанная Карлом Тейтом (Karl Tate), с помощью инфографики рассказывает о том, как продвинутая цивилизация могла бы получать энергию звезды, используя огромные массивы солнечных панелей.

«Ringworld»[1] (Ballantine Books, 1970), написанная Ларри Нивеном (Larry Niven), — классический научно-фантастический роман о полете к массивной заброшенной инопланетной конструкции — Миру-Кольцу, сооруженной вокруг далекой звезды.

Практический проект: обнаружение инопланетных мегаструктур

В 2015 году ученые, анализирующие данные с космического телескопа Kepler, заметили кое-что странное возле звезды Табби, расположенной в созвездии Лебедя. Кривая блеска звезды, зарегистрированная в 2013 году, демонстрировала нерегулярные изменения яркости, которые казались слишком большими, чтобы быть вызванными планетой (рис. 8.17).

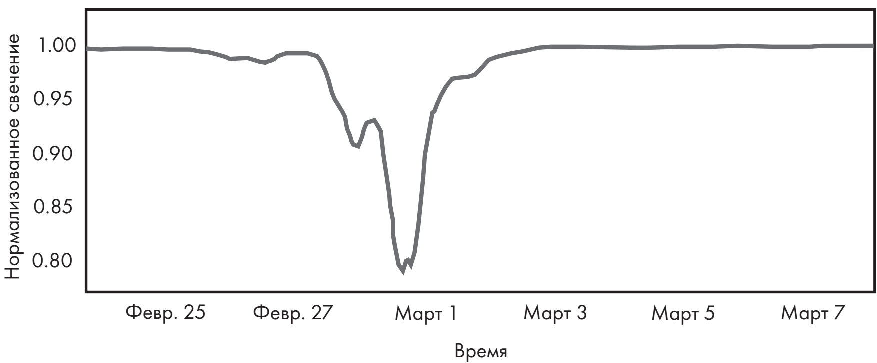


Рис. 8.17. Кривая блеска звезды Табби, измеренная космической обсерваторией Kepler

Помимо этого, кривая блеска была несимметрична и включала странные неровности, которые не регистрируются при типичных транзитах планет. Объяснить это явление пытались по-разному: поглощением планеты звездой, транзитом облака распадающихся комет, наличием большой окруженной кольцами планеты, сопровождаемой скоплениями астероидов, а также существованием *инопланетной мегаструктуры*.

[1] Нивен Л. «Мир-Кольцо».

Ученые высказали догадку, что искусственная конструкция подобного размера с наибольшей вероятностью может быть попыткой инопланетной цивилизации аккумулировать энергию от своей звезды. Подобные крупномасштабные проекты построения солнечных панелей описаны как в научной, так и в фантастической литературе: рой Дайсона, сфера Дайсона, Мир-Кольцо и раковина Покровского (рис. 8.18).

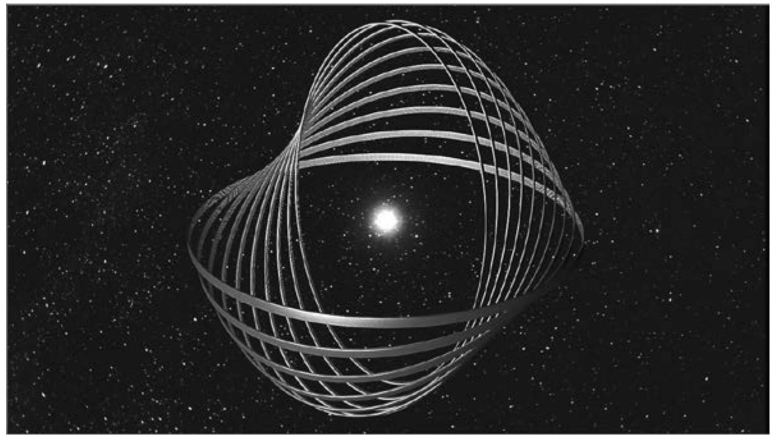


Рис. 8.18. Раковина Покровского — совокупность колец, спроектированных вокруг звезды для перехвата ее излучения

В этом практическом проекте используйте программу transit.py, чтобы аппроксимировать форму и глубину кривой блеска звезды Табби. Замените используемую в программе круглую экзопланету на другие простые геометрические фигуры. От вас не требуется воссоздать кривую в точности; просто проанализируйте ее ключевые признаки, такие как асимметрия, выпуклость в виде «бугорка» в районе 28 февраля, и большой провал в яркости.

Мой вариант под названием practice_tabbys_star.py вы найдете в каталоге Chapter_8, доступном для скачивания с сайта книги по адресу https://nostarch.com/real-world-python/https://nostarch.com/real-world-python/, а также в приложении. На рис. 8.19 показано, какая кривая блеска получается.

Нам известно, что независимо от того, что именно вращается вокруг звезды Табби, оно пропускает волны света определенной длины, значит, это не может быть твердым объектом. Исходя из этого предположения и длин волн, поглощаемых объектом, ученые считают, что в изменении кривой блеска звезды виновата космическая пыль. Однако другие звезды, например HD 139139 в созвездии Весов, тоже демонстрируют странные кривые блеска, которые на момент написания книги все еще не удается объяснить.

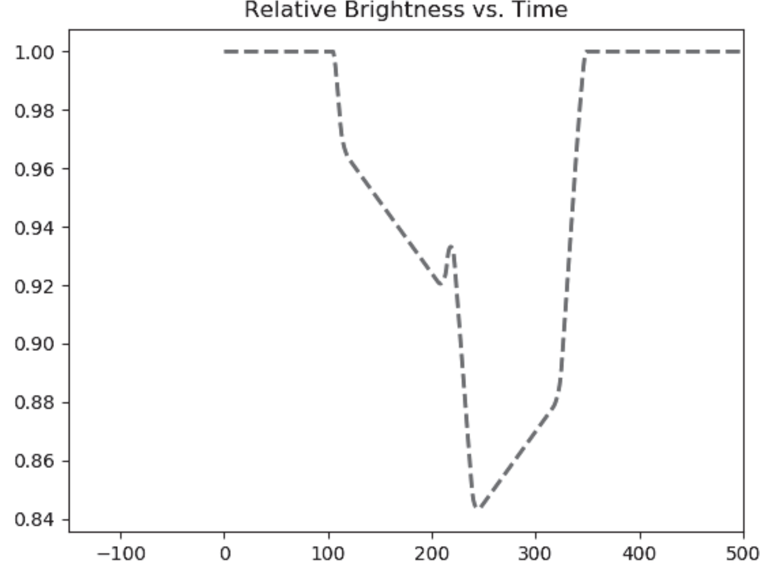


Рис. 8.19. Кривая блеска, создаваемая программой practice_tabbys_star.py

Практический проект: обнаружение транзита астероидов

Поля астероидов могут также обусловливать неровные и асимметричные кривые блеска. Эти пояса космических обломков зачастую формируются при столкновении планет или при рождении солнечных систем, например Троянские астероиды на орбите Юпитера (рис. 8.20). Интересную анимацию на эту тему вы найдете на странице «Lucy: The First Mission to the Trojan Asteroids» по адресу https://www.nasa.gov/.

Измените программу transit.py, чтобы она случайным образом создавала астероиды с радиусом от 1 до 3, с преобладанием радиусов, близких к 1. Пусть пользователь вводит их количество. Не утруждайте себя вычислением радиуса экзопланеты, поскольку эти расчеты предполагают работу с одним сферическим объектом. Поэкспериментируйте с количеством астероидов, их размерами и распределением (диапазон x и диапазон y, в котором они существуют), чтобы оценить оказываемое ими влияние на кривую блеска. Один из примеров показан на рис. 8.21.

Решение под названием practice_asteroids.py вы найдете в приложении к книге, а также на ее сайте. В этой программе используется объектно-ориентированное

программирование (ООП) для упрощения управления большим числом астероидов.

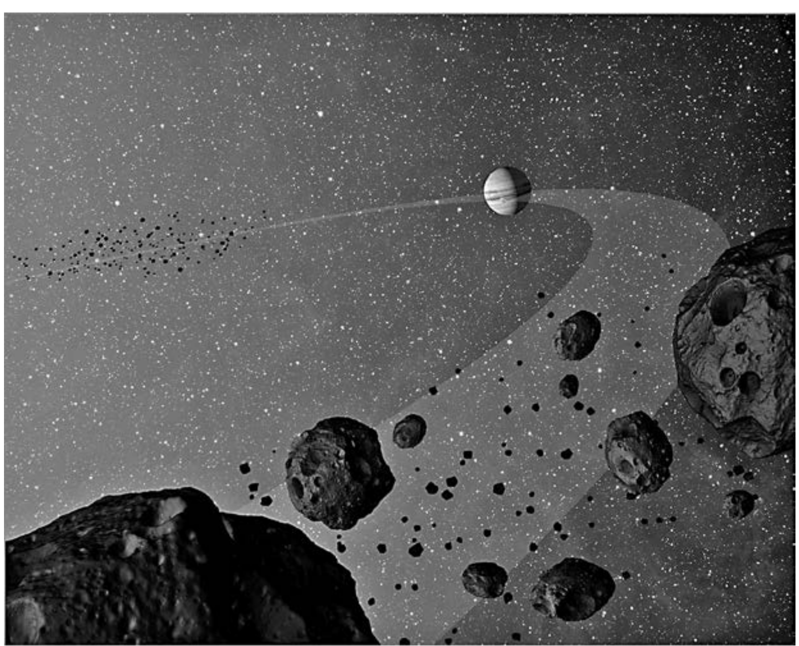


Рис. 8.20. На орбите Юпитера вращается более миллиона Троянских астероидов

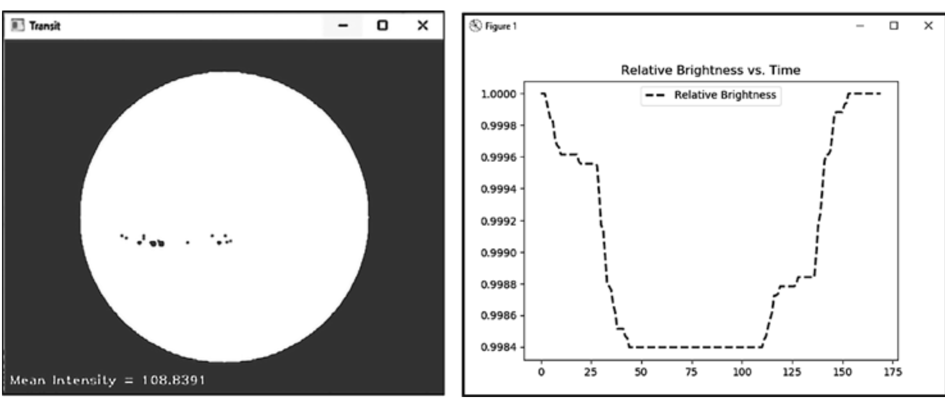


Рис. 8.21. Нерегулярная асимметричная кривая блеска, произведенная случайно сгенерированным полем астероидов

Практический проект: добавление эффекта потемнения к краю

Фотосфера — это внешний светящийся слой звезды, который излучает свет и тепло. Поскольку с увеличением расстояния от центра звезды температура фотосферы падает, края диска оказываются менее нагретыми и по сравнению с центром звезды выглядят более тускло (рис. 8.22). Этот эффект называется *потемнением диска к краю*.

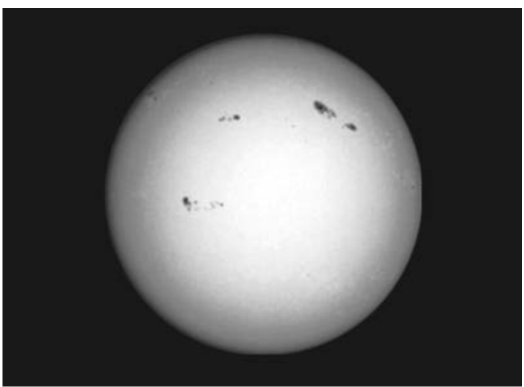


Рис. 8.22. Потемнение солнечного диска к краю и пятна на Солнце

Перепишите программу transit.py, чтобы в ней учитывалось потемнение диска к краю. Вместо того чтобы рисовать звезду, используйте картинку limb_darlening.png из каталога Chapter_8, который можно скачать с сайта книги.

Потемнение к краю влияет на кривые блеска, демонстрирующие транзит планет. По сравнению с теоретическими кривыми, которые мы создавали в проекте 11, они будут получаться менее прямоугольными, с более закругленными, мягкими краями и изогнутым дном (рис. 8.23).

Используя обновленную версию программы, вернитесь к разделу «Эксперименты с транзитной фотометрией» на с. 242, где мы анализировали кривую блеска при частичных транзитах. Вы увидите, что по сравнению с частичными полные транзиты по-прежнему демонстрируют более широкие провалы с плоским дном (рис. 8.24).

Если полный транзит планеты меньшего радиуса произойдет возле края звезды, ввиду потемнения к краю отличить его от частичного транзита более крупной планеты будет сложно. Это можно наблюдать на рис. 8.25, где стрелками указано расположение планет.

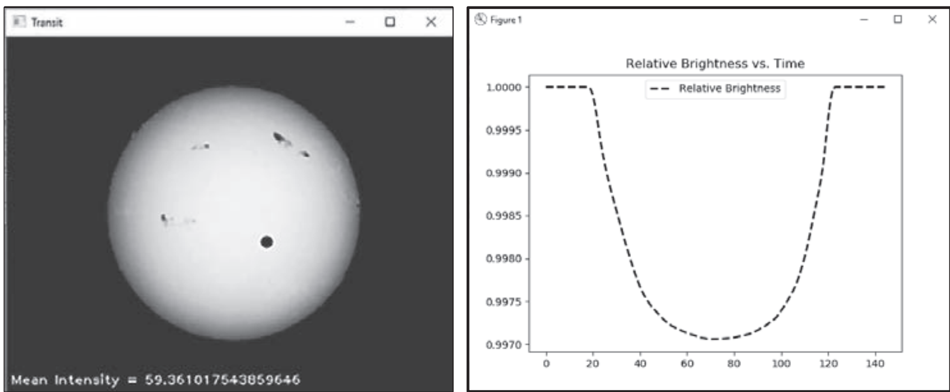


Рис. 8.23. Эффект, оказываемый потемнением к краю на кривую блеска

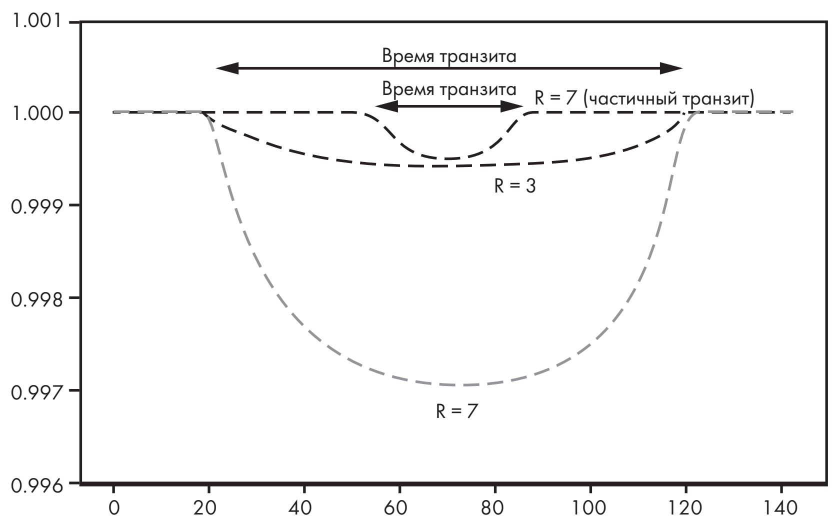


Рис. 8.24. Кривые блеска с учетом потемнения к краю для полного и частичного транзитов (R = радиус экзопланеты)

Астрономы могут извлечь множество информации, анализируя кривую блеска. С помощью регистрации нескольких событий транзита они определяют орбитальные параметры экзопланеты, такие как расстояние между планетой и звездой. На основе малейших изгибов кривой блеска ученые рассчитывают интервал времени, в течение которого планета полностью находится на фоне звезды. Также они могут теоретически оценить величину потемнения к краю и использовать моделирование, как мы с вами сейчас, чтобы собрать все данные воедино и протестировать свои предположения в отношении реальных наблюдений.

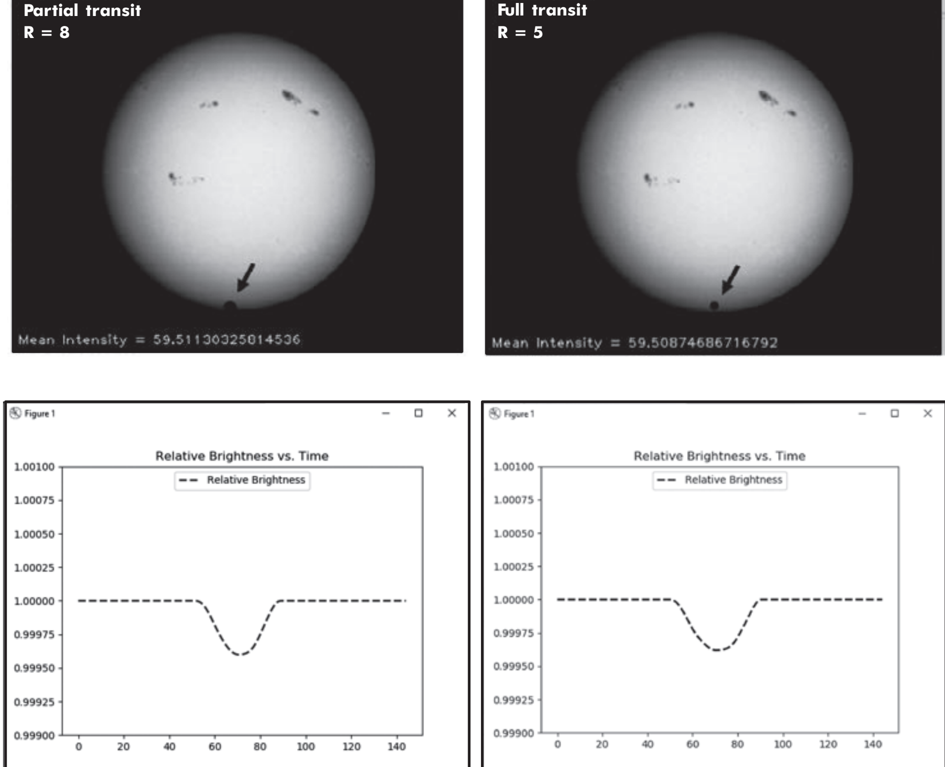


Рис. 8.25. Частичный транзит планеты с радиусом 8 пикселей в сравнении с полным транзитом планеты с радиусом 5 пикселей

Решение под названием practice_limb_darkening.py вы найдете в приложении и в каталоге Chapter_8, доступном для скачивания на сайте книги.

Практический проект: обнаружение пятен на звездах

Солнечные пятна — на других солнцах их именуют звездными — это области поверхности со сниженной температурой, появление которых вызвано изменениями в магнитном поле звезды. Звездные пятна могут затемнять лицевую сторону звезд и интересным образом влиять на кривые их блеска. На рис. 8.26 экзопланета проходит над звездным пятном, вследствие чего на кривой возникает резкий пик.

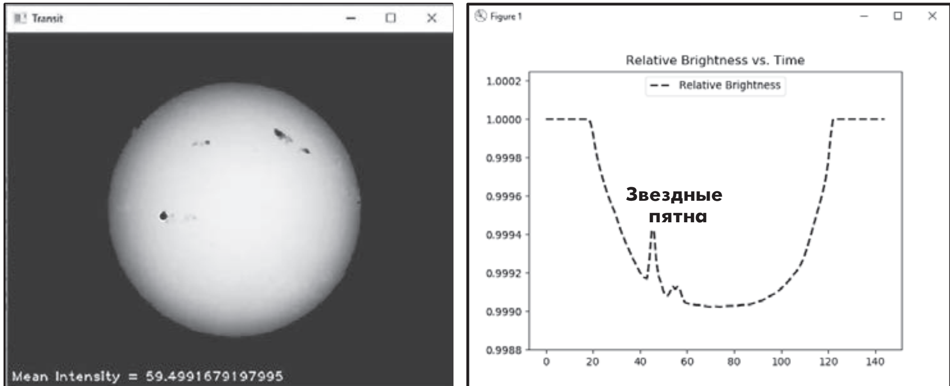


Рис. 8.26. Экзопланета (указана стрелкой на левом изображении), проходящая над звездным пятном, создает выпуклость на кривой блеска

Чтобы поэкспериментировать со звездными пятнами, используйте код practice_limb_darkening.py из предыдущего практического проекта, отредактировав его так, чтобы экзопланета примерно одного размера со звездными пятнами проходила над ними при транзите. Для воссоздания рис. 8.26 используйте значения `EXO_RADIUS = 4`, `EXO_DX = 3` и `EXO_START_Y = 205`.

Практический проект: обнаружение инопланетной армады

Гиперэволюционировавшие бобры экзопланеты BR549 были трудягами, впрочем, как и любые другие бобры. Они собирали армаду исполинских кораблей, чтобы покинуть орбиту своей опустошенной планеты. Бобры обнаружили экзопланету, похожую по свойствам на их собственную, и решили переселиться в густые зеленые леса Земли!

Напишите программу Python, симулирующую траекторию движения нескольких звездных кораблей, которые совершают транзит мимо звезды. Задайте для этих кораблей разный размер, форму и скорость (как на рис. 8.27).

Сравните полученную кривую блеска с ее вариантом для звезды Табби (см. рис. 8.17) и с той, что получена в проекте по обнаружению астероидов. Отличаются ли кривые для кораблей или же аналогичные кривые можно получить для скоплений астероидов, звездных пятен и других естественных явлений?

Решение под названием practice_alien_armada.py вы найдете в приложении, а также в каталоге Chapter_8, доступном для скачивания с сайта книги.

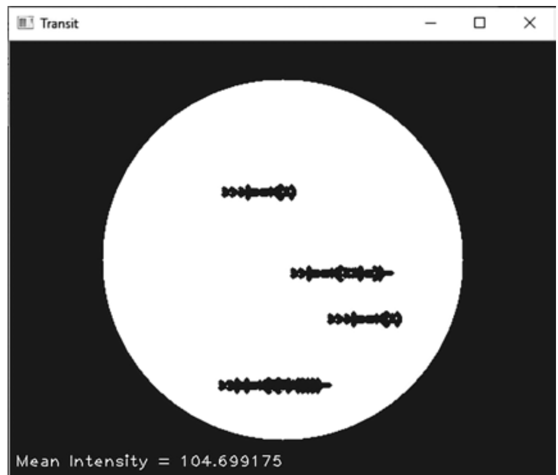


Рис. 8.27. Армада кораблей инопланетян готовится к вторжению на Землю

Практический проект: обнаружение планеты с Луной

Какую кривую блеска дала бы экзопланета с вращающейся вокруг нее Луной? Напишите программу Python, симулирующую небольшую экзолуну, вращающуюся вокруг более крупной экзопланеты, и вычислите итоговую кривую блеска. Решение называется practice_planet_moon.py и находится в приложении к книге или на сайте.

Практический проект: измерение продолжительности экзопланетного дня

Старший астроном предоставил вам 34 изображения экзопланеты BR549, полученные час назад. Напишите программу Python, которая загружает изображения по порядку, измеряет интенсивность яркости для каждого и строит для всех одну кривую блеска (рис. 8.28). Используйте эту кривую для определения продолжительности дня на BR549.

Решение под названием practice_length_of_day.py находится в приложении. Цифровая же версия его кода вместе с каталогом изображений (br549_pixelated) — в каталоге Chapter_8, доступном для скачивания на сайте книги.

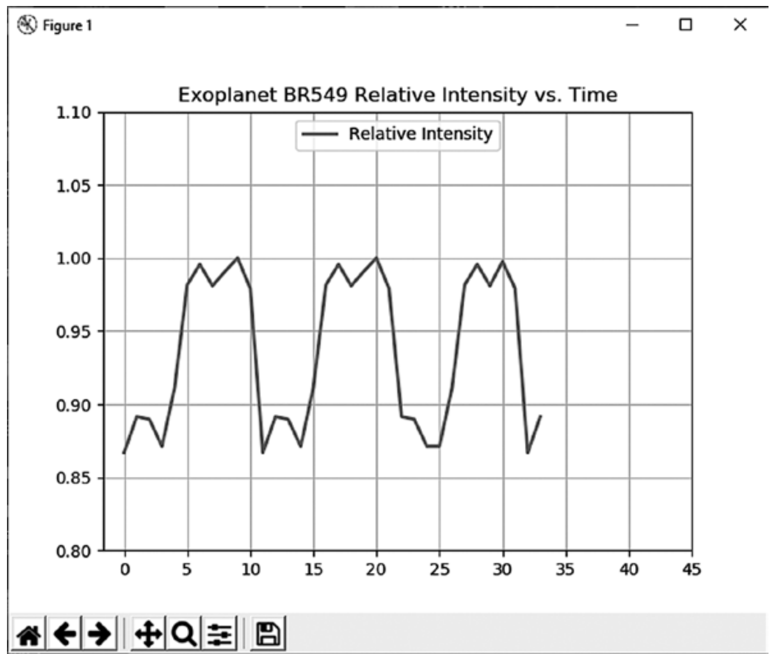


Рис. 8.28. Общая кривая блеска на основе 34 изображений экзопланеты BR549

Усложняем проект: генерация динамической кривой блеска

Перепишите transit.py так, чтобы кривая блеска появлялась не по завершении симуляции, а динамически обновлялась по ходу ее выполнения.

9

Как различить своих и чужих

Обнаружение лиц — это технология машинного обучения (МО), которая находит человеческие лица на цифровых изображениях. Это первый этап еще одного метода МО — *распознавания* отдельных лиц. Методы обнаружения и распознавания человеческих лиц широко применяются: например, это разметка фотографий в социальных сетях, автофокусировка цифровых камер, разблокировка сотовых телефонов, поиск потерявшихся детей, отслеживание террористов, повышение безопасности платежей и многое другое.

В текущей главе мы используем алгоритмы МО в OpenCV, чтобы запрограммировать робота-часового. И поскольку отличать нужно будет людей от инопланетных мутантов, то потребуется лишь обнаруживать *наличие* на изображениях человеческих лиц, но не распознавать конкретных людей. А вот в главе 10 мы уже займемся распознаванием людей по их лицам.

Обнаружение лиц на фотографиях

Обнаружение лиц возможно благодаря тому, что лица обладают наборами схожих характеристик, или паттернов. К таким характеристикам относятся, например, более темный цвет глаз по сравнению со щеками и более светлый цвет переносицы в сравнении с глазами, как показано на рис. 9.1.

Паттерны можно извлекать, используя шаблоны, подобные приведенным на рис. 9.2. В результате мы получаем так называемые *признаки Хаара*. Это

атрибуты цифровых изображений, используемые в распознавании объектов. Для получения признака Хаара нужно поместить один из шаблонов на полутоновое изображение, сложить полутоновые пиксели, попадающие под белую часть шаблона, и вычесть их из суммы пикселей, попадающих под черную. Таким образом, каждый признак состоит из одного значения интенсивности. При этом можно использовать различные размеры шаблонов для оценки всех возможных зон изображения, делая систему нечувствительной к масштабу.

Рис. 9.1. Примеры некоторых светлых и темных участков лица

Рис. 9.2. Пример шаблонов признаков Хаара

В середине изображения на рис. 9.1 шаблон «граничного признака» демонстрирует связь между темными глазами и светлыми щеками. На крайнем правом изображении того же рисунка шаблон «линейного признака» показывает связь между темными глазами и светлым носом.

Путем вычисления признаков Хаара для тысяч *известных* изображений лиц и нелиц можно определить, какая комбинация этих признаков окажется наиболее эффективной для идентификации лиц. Процесс такого обучения очень медленный, но при работе он существенно упрощает детекцию. Создаваемый подобным образом алгоритм, *классификатор лиц*, получает значения признаков на изображении и прогнозирует, находится ли на нем человеческое лицо, выводя 1 либо 0. OpenCV изначально содержит предварительно обученный классификатор для детекции лиц, основанный на этой технике.

Для его применения алгоритм использует технику *скользящего окна*. Небольшая прямоугольная область инкрементно перемещается вдоль изображения и оценивается при помощи *каскадного классификатора*, состоящего из нескольких

этапов фильтрации. Фильтры каждого этапа представляют собой комбинации признаков Хаара. Если область окна не может пройти пороговое значение стадии, она отвергается и окно сдвигается в следующую позицию. Быстрое отклонение нелицевых областей, подобных показанному на правом сегменте рис. 9.3, ускоряет процесс.

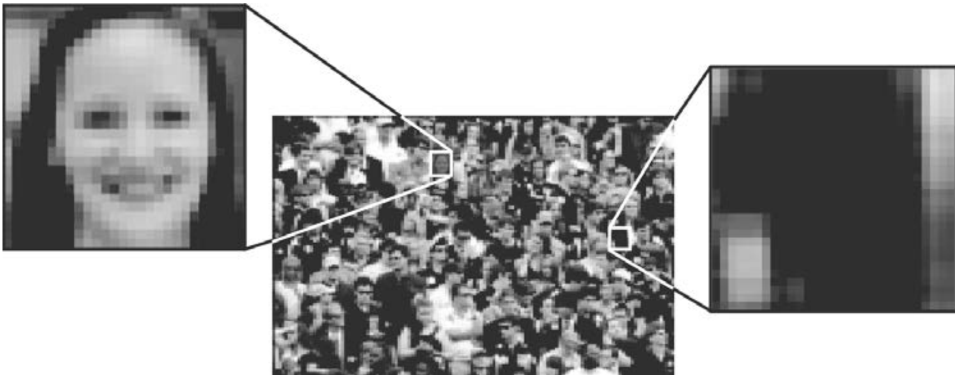


Рис. 9.3. На изображениях выполняется поиск лиц при помощи скользящего окна

Если область преодолевает установленный для стадии порог, алгоритм переходит к обработке другого набора признаков Хаара, после чего снова сравнивает их с порогом и так далее, пока либо не отвергнет, либо не подтвердит присутствие лица. В результате такого процесса скорость движения окна вдоль изображения то ускоряется, то замедляется. Отличный видеопример вы найдете по ссылке https://vimeo.com/12774628/.

Для каждого обнаруженного лица алгоритм возвращает координаты окружающего его прямоугольника. Эти прямоугольники можно использовать в качестве основы для дальнейшего анализа.

Проект #13. Программирование робота-часового

Представьте, что вы служите техником в пехоте Коалиции, являющейся частью Звездных сил. Ваш взвод расположился на секретной исследовательской базе под управлением Wykham-Yutasaki Corporation на планете LV-666. В ходе изучения загадочного инопланетного механизма исследователи случайно открыли портал в инфернальное измерение. Все, кто оказался рядом

с этим порталом, включая десятки гражданских лиц и нескольких из ваших сослуживцев, мутировали в злобных безумных монстров! Вы даже получили снимки камеры наблюдения, где можно наглядно видеть результат мутации (рис. 9.4).

Рис. 9.4. Снятые на камеру наблюдения мутировавший ученый (*слева*) и пехотинец (*справа*)

Со слов выживших ученых, мутация влияет не только на органическую материю. Любые предметы, имеющиеся при жертве, включая шлемы и очки, трансформируются и сливаются с плотью. Глазная ткань особенно уязвима. Все возникшие на данный момент мутанты лишены глаз и, следовательно, слепы, хотя на их подвижность и ориентацию это никак не влияет. Они попрежнему свирепы, смертоносны и неостановимы без помощи серьезного боевого оружия.

Теперь дело за вами. Ваша задача — настроить автоматическую огнестрельную турель, которая будет охранять коридор 5, ключевую точку доступа в помещение, где произошел инцидент. Без такой меры защиты ваш небольшой взвод будет окружен и уничтожен ордами свирепых мутантов.

Огнестрельная платформа состоит из автоматической самонаводящейся турели UAC-549, которую пехотинцы попросту зовут *роботом-часовым* (рис. 9.5). Она оборудована автоматическими пушками M30 с 1000 снарядов, а также несколькими сенсорами, включая датчик движения, лазерный дальномер и оптическую камеру. Турель также идентифицирует обнаруженные цели с помощью транспондера, работающего по принципу «свой/чужой» (IFF, identification friend or foe). Все пехотинцы носят такие транспондеры, что позволяет им безопасно проходить активные боевые турели.

Рис. 9.5. Автоматическая турель UAC 549-B

К сожалению, при посадке на планету оборудование турели немного пострадало, поэтому транспондеры больше не работают. Еще хуже, что каптенармус забыл скачать ПО для визуального распознавания целей. Теперь, когда сенсор транспондеров вышел из строя, возможности распознать десантников и мирных жителей нет. Вам надо исправить это как можно скорее, потому что ваши силы малы, а мутанты уже наступают.

К счастью, на планете LV-666 нет аборигенов, поэтому вам придется различать только людей и мутантов. Поскольку мутанты, по сути, безликие, логично создать алгоритм обнаружения лиц.

ЗАДАЧА

Написать программу на Python, деактивирующую боевой механизм турели, если на изображении обнаружено человеческое лицо.

Стратегия

В подобных ситуациях лучше делать все максимально просто и применить уже имеющиеся ресурсы. Это означает — положиться на встроенную в OpenCV функциональность обнаружения лиц вместо того, чтобы писать собственный код. Однако мы не можем быть уверены в стопроцентной правильности работы этих стандартных процедур, поэтому потребуется давать подсказки, чтобы максимально упростить задачу.

Находящийся на турели детектор движения будет обрабатывать функцию активации процесса оптической идентификации. Чтобы люди могли проходить беспрепятственно, нужно предупреждать их о необходимости остановиться и посмотреть в камеру. Эта процедура потребует нескольких секунд, затем при успешном распознавании человек сможет двигаться дальше.

Нам также придется провести кое-какие тесты, чтобы гарантировать адекватность обучающей выборки OpenCV и отсутствие ложноположительных срабатываний, которые позволят мутантам пройти проверку. При этом мы не хотим по случайности убить кого-то из своих, но и чересчур осторожничать в этом вопросе тоже нельзя. Если один мутант вдруг проберется, то пострадают многие.

ПРИМЕЧАНИЕ

В реальности автоматическая турель использовала бы видеопоток, но так как у меня нет собственной киностудии со спецэффектами и гримерами, то мы будем работать с фотографиями, рассматривая их как отдельные кадры видео. Позже в этой главе вы попробуете обнаружить собственное лицо, используя камеру вашего компьютера.

Код

Код sentry.py перебирает каталог изображений, определяет на них человеческие лица и показывает изображение, где обнаруженные лица обведены рамками. После этого в зависимости от результата пушка либо стреляет, либо нет. Мы используем изображения из директории corridor_5 каталога Chapter_9, доступного для скачивания с https://nostarch.com/real-world-python/. Как всегда, не перемещайте и не переименовывайте никакие файлы после их скачивания и запускайте sentry.py из каталога, в котором он хранится.

Вам также потребуется установить два модуля, `playsound` и `pyttsx3`. Первый — это кроссплатформенный модуль для проигрывания WAV- и MP3-файлов.

С его помощью мы зададим звуковые эффекты, такие как стрельба пулеметов и звук «all clear» (сигнал отбоя). Второй же модуль — кроссплатформенная обертка, которая поддерживает нативные библиотеки синтеза речи по тексту

для систем Windows, Linux, а также macOS. Мы используем его для озвучивания предупреждений и инструкций, которые выдает автоматическая турель. В отличие от других библиотек синтеза речи, `pyttsx3` считывает текст напрямую из программы, не сохраняя его сначала в аудиофайл. Помимо этого, она работает офлайн, что более надежно для проектов, где используется голос.

Оба модуля можно установить с помощью pip в окне PowerShell или терминала.

```
pip install playsound
pip install pyttsx3
```

Если при установке `pyttsx3` в Windows возникнет ошибка, например `No module named win32.com`, `No module named win32` или `No module named win32api`, тогда установите `pypiwin32`.

```
pip install pypiwin32
```

После установки вам может потребоваться перезапустить оболочку Python и редактор.

Подробности относительно `playsound` ищите на странице https://pypi.org/project/playsound/. Документацию для `pyttsx3` вы найдете на страницах https://pyttsx3.readthedocs.io/en/latest/ и https://pypi.org/project/pyttsx3/.

Если у вас еще не установлена OpenCV, обратитесь к разделу «Установка библиотек Python» на с. 31.

Импорт модулей, настройка аудио, а также обращение к файлам классификатора и изображениям коридора

Код листинга 9.1 импортирует модули, инициализирует и настраивает аудиодвижок, присваивает файлы классификатора переменным и изменяет текущий каталог на тот, где содержатся изображения коридора.

Листинг 9.1. Импорт модулей, настройка аудио, а также обнаружение файлов классификатора и изображений коридора

sentry.py, part 1

```
   import os
   import time
❶  from datetime import datetime
   from playsound import playsound
   import pyttsx3
   import cv2 as cv

❷  engine = pyttsx3.init()
   engine.setProperty('rate', 145)
```

```
   engine.setProperty('volume', 1.0)

   root_dir = os.path.abspath('.')
   gunfire_path = os.path.join(root_dir, 'gunfire.wav')
   tone_path = os.path.join(root_dir, 'tone.wav')

❸  path= "C:/Python372/Lib/site-packages/cv2/data/"
   face_cascade = cv.CascadeClassifier(path +
                                   'haarcascade_frontalface_default.xml')
   eye_cascade = cv.CascadeClassifier(path + 'haarcascade_eye.xml')

❹  os.chdir('corridor_5')
   contents = sorted(os.listdir())
```

Если вы проработали материал предыдущих глав, то вам знакомы все модули ❶, за исключением `datetime`, `playsound` и `pytts3`. Модуль `datetime` мы задействуем для записи точного времени обнаружения вошедшего в коридор объекта.

Чтобы использовать `pytts3`, инициализируем объект `pyttsx3` и присваиваем его переменной, по соглашению названной `engine` ❷. Согласно документации `pyttsx3`, приложение использует объект `engine` для регистрации и отмены регистрации обратных вызовов события, воспроизведения и остановки речи, получения и установки свойств движка речи, а также начала и остановки циклов событий.

В следующих двух строках устанавливаем свойства скорости речи и громкости. Используемое здесь значение скорости речи было получено экспериментальным путем. Она должна быть быстрой, но при этом оставаться четкой и понятной. Громкость следует установить на максимальное значение (`1.0`), чтобы любой человек в коридоре мог легко услышать предупреждение.

По умолчанию в Windows используется мужской голос, но можно выбрать и другие варианты. К примеру, на машине с Windows 10 можно переключиться на женский, используя следующий голосовой ID:

```
engine.setProperty('voice',
'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Speech\Voices\Tokens\TTS_MS_EN-US_ZIRA_11.0')
```

Чтобы увидеть список доступных на вашей платформе голосов, обратитесь к разделу «Changing voices» на странице https://pyttsx3.readthedocs.io/en/latest/.

Далее настраиваем аудиозапись стрельбы, которая будет воспроизводиться при обнаружении в коридоре мутанта. Укажите расположение аудиофайла, сгенерировав строку пути для каталога — он будет работать на всех платформах. Для этого нужно совместить абсолютный путь с именем файла при помощи метода `os.path.join()`. Тот же путь используйте для файла tone.wav, который выдает сигнал «all clear» при обнаружении программой человека.

Предварительно обученные каскадные классификаторы Хаара должны быть скачаны в виде .xml-файлов при установке OpenCV. Присваиваем путь для каталога с классификаторами переменной ❸. Здесь вы видите путь для моей машины с Windows. В вашем случае путь может отличаться. К примеру, в macOS они располагаются в opencv/data/haarcascades. Эти классификаторы также можно найти онлайн по адресу https://githb.com/opencv/opencv/tree/master/data/haarcascades/.

Еще один вариант найти путь к каскадным классификаторам — использовать предварительно установленный модуль `sysconfig`, как показано в этом сниппете:

```
>>> import sysconfig
>>> path = sysconfig.get_paths()['purelib'] + '/cv2/data'
>>> path
'C:\\Python372\\Lib\\site-packages/cv2/data'
```

Это должно сработать для Windows как вне, так и внутри виртуальных сред. Однако в Ubuntu такой способ сработает только в последнем случае.

Загружаем классификатор, используя метод OpenCV `CascadeClassifier()`. С помощью конкатенации строк добавляем переменную пути к строке имени файла классификатора и присваиваем результат переменной.

Обратите внимание, что с целью упрощения я использую только два классификатора: один для фронтальных позиций лиц и второй для глаз. При этом также есть классификаторы для вида в профиль, улыбок, очков, туловища и т. д.

Завершаем код, указывая программе на изображения охраняемого коридора. Изменяем соответствующим образом каталог ❹, затем перечисляем его содержимое и присваиваем результаты переменной `contents`. Так как мы не предоставляем полный путь к каталогу, нужно запускать программу из содержащей ее папки, которая должна находиться на один уровень выше каталога с изображениями.

Озвучивание предупреждения, загрузка изображений и обнаружение лиц

Листинг 9.2 начинает цикл `for` для перебора каталога, содержащего изображения коридора. В реальности датчики движения турели запускали бы программу сразу при входе кого-либо в коридор. Но так как у нас датчиков движения нет, мы предположим, что каждый цикл представляет появление объекта в коридоре.

Цикл сразу же активирует пушку для стрельбы. После этого вошедшему объекту голосом предлагается остановиться и посмотреть в камеру. Это осуществляется на заданном расстоянии от пушки, как если бы срабатывал датчик движения. В результате лица будут получаться все примерно одного размера, что упростит тестирование программы.

Вошедшему дается пять секунд на то, чтобы выполнить команду. После этого вызывается каскадный классификатор, который осуществляет поиск лиц.

Листинг 9.2. Перебираем изображения, отправляя звуковое предупреждение и выполняя поиск лиц

sentry.py, часть 2

```
for image in contents:
  ❶ print(f"\nMotion detected...{datetime.now()}") # Обнаружено движение
    discharge_weapon = True
  ❷ engine.say("You have entered an active fire zone. \
                 Stop and face the gun immediately. \
                 When you hear the tone, you have 5 seconds to pass.")
    engine.runAndWait()
    time.sleep(3)

  ❸ img_gray = cv.imread(image, cv.IMREAD_GRAYSCALE)
    height, width = img_gray.shape
    cv.imshow(f'Motion detected {image}', img_gray)
    cv.waitKey(2000)
    cv.destroyWindow(f'Motion detected {image}')

  ❹ face_rect_list = []
    face_rect_list.append(face_cascade.detectMultiScale(image=img_gray,
                                                         scaleFactor=1.1,
                                                         minNeighbors=5))[1]
```

Начинаем перебор изображений в каталоге. Каждый новый образец представляет следующего вошедшего в коридор. Выводим журнал события и время, когда оно произошло ❶. Обратите внимание на `f` перед началом строки. Это формат `f`-строк, появившийся в Python 3.6 (https://www.python.org/dev/peps/pep-0498/). Это строковый литерал, в фигурных скобках которого содержатся переменные, строки, математические операции и даже вызовы функций. Когда программа выводит строку, она заменяет эти выражения их значениями. Это самый быстрый и наиболее эффективный формат строк в Python, а нам определенно нужно добиться от программы максимальной скорости.

Предположим, что каждый вошедший является мутантом. Вербально предупреждаем о необходимости остановиться и пройти сканирование.

Для озвучивания речи используем метод `say()` объекта `engine` ❷. Он получает в качестве аргумента строку. Сопровождаем его методом `runAndWait()`, который приостановит программу, очистит очередь `say()` и воспроизведет аудио.

[1] Перевод текста аудио: «Вы вошли в зону огня. Немедленно остановитесь и повернитесь лицом к оружию. После сигнала у вас будет 5 секунд, чтобы пройти».

ПРИМЕЧАНИЕ

У некоторых пользователей macOS программа может завершиться при втором вызове runAndWait(). Если это произойдет, скачайте код sentry_for_Mac_bug.py с сайта книги. Эта программа использует вместо pyttsx3 функциональность синтеза речи по тексту операционной системы. Вам также понадобится обновить переменную пути классификаторов Хаара, как мы делали в блоке ❸ листинга 9.1.

Далее с помощью модуля `time` приостанавливаем программу на три секунды, давая вошедшему время встать лицом к камере турели.

В этот момент должен происходить захват видео, но мы работаем не с видео. Вместо этого загружаем изображения из каталога corridor_5, для чего вызываем метод `cv.imread()` с флагом `IMREAD_GRAYSCALE` ❸.

Используя атрибут `shape` изображения, мы получаем его высоту и ширину в пикселях. Это пригодится позже при размещении на изображениях текста.

Обнаружение лиц работает только для полутоновых изображений, но OpenCV внутренне конвертирует их цветные версии при применении каскадов Хаара. Я решил изначально задействовать полутоновые изображения, поскольку их результаты при отображении выглядят более жутко. Если же вы хотите использовать цветные, то просто измените две предыдущие строки таким образом:

```
img_gray = cv.imread(image)
height, width = img_gray.shape[:2]
```

Теперь показываем изображение до обнаружения лиц, задерживаем его на две секунды (ввод в миллисекундах), а затем закрываем окно. Это необходимо для контроля качества, чтобы убедиться в выполнении проверки всех изображений. После того как станет ясно, что все работает отлично, эти шаги можно будет закомментировать.

Создаем пустой список для хранения всех лиц, обнаруженных на текущем изображении ❹. OpenCV рассматривает изображения как массивы `NumPy`, поэтому элементы в этом списке представляют координаты (x, y, ширина, высота) угловых точек прямоугольника, обрамляющего лицо, как показано в следующем фрагменте вывода:

```
[array([[383, 169, 54, 54]], dtype=int32)]
```

Теперь обнаруживаем лица с помощью каскадов Хаара. Это мы делаем для переменной `face_cascade` через вызов метода `detectMultiscale()`. Методу передается изображение, а также значения для фактора масштабирования

и минимального количества соседей. Их можно использовать для настройки результатов в случае ложноположительного срабатывания или неудачи при распознавании лиц.

Для получения хороших результатов лица на изображении должны по размеру совпадать с теми, которые использовались для обучения классификатора. Для этого параметр `scaleFactor` изменяет масштаб оригинального изображения до корректного размера, используя технику под названием *«пирамида изображений»* (рис. 9.6).

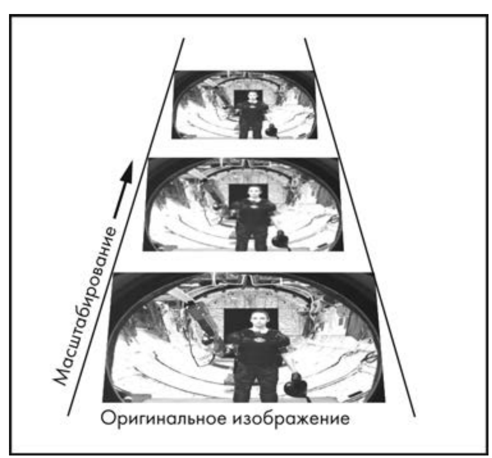


Рис. 9.6. Пример пирамиды изображений

Пирамида изображений поэтапно уменьшает размер изображения заданное количество раз. К примеру, значение `1.2` для `scaleFactor` означает, что картинка будет уменьшаться с шагом 20 %. Скользящее окно будет двигаться вдоль получившегося уменьшенного изображения и повторять проверку на наличие признаков Хаара. Это сжатие и скольжение продолжается, пока масштабированное изображение не совпадет по размеру с тем, которое использовалось для обучения. В каскадном классификаторе Хаара этот размер составляет 20 × 20 пикселей (можете убедиться, открыв один из файлов.xml). Окна меньшего размера обнаружить не удается, поэтому дальше размер не уменьшается. Обратите внимание, что пирамида изображений будет только *уменьшать* их, так как увеличение может внести артефакты.

С каждым этапом изменения масштаба алгоритм вычисляет множество новых признаков Хаара, в результате чего получается большое число ложноположительных срабатываний. Для их исключения используем параметр `minNeighbors`.

Чтобы увидеть процесс в действии, взгляните на рис. 9.7. На нем прямоугольники обозначают лица, обнаруженные классификатором `haarcascade_frontalface_alt2.xml` при параметре `scaleFactor`, установленном на `1.05`, а `minNeighbors` на `0`. Размеры прямоугольников различаются, это зависит от того, какого масштаба изображение — согласно параметру `scaleFactor` — обрабатывалось при обнаружении лица. Несмотря на большое число ложноположительных результатов, прямоугольники в основном накапливаются вокруг реального лица.

Рис. 9.7. Прямоугольники вокруг обнаруженных лиц при minNeighbors=0

Увеличение значения параметра `minNeighbors` приведет к увеличению качества обнаружений, но уменьшит их количество. Если указать значение 1, то будут сохранены только те прямоугольники, рядом с которыми находится не менее одного прямоугольника. Все остальные при этом будут удалены (рис. 9.8).

Увеличение минимального числа соседей до пяти (ориентировочно) обычно исключает ложные срабатывания (рис. 9.9). Этого может быть достаточно для большинства задач, но в случае с ужасными монстрами из другого измерения необходимы дополнительные ограничения.

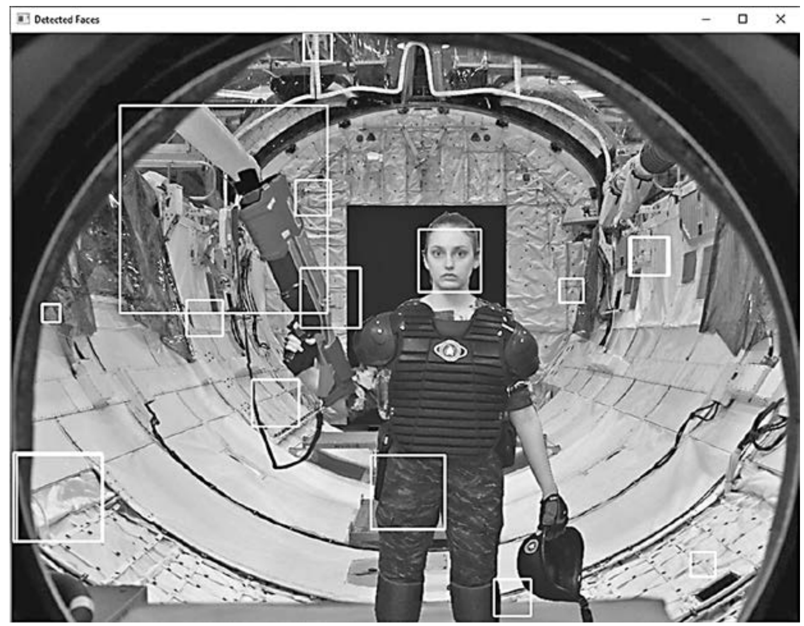


Рис. 9.8. Прямоугольники вокруг обнаруженных лиц при minNeighbors=1

Рис. 9.9. Прямоугольники вокруг обнаруженных лиц при minNeighbors=5

Чтобы понять почему, взгляните на рис. 9.10. Несмотря на использование для `minNeighbor` значения 5, область большого пальца ноги мутанта ошибочно распознана как лицо. Если включить воображение, то можно разглядеть два темных глаза и светлый нос в верхней части этого прямоугольника, а также темный прямой рот у основания. В итоге такой проверки мутанту удалось бы пройти невредимым, в результате вас в лучшем случае разжаловали бы, а в худшем — вы погибли бы мучительной смертью.

Рис. 9.10. Большой палец правой ноги мутанта опознан как лицо

К счастью, эту проблему можно легко исправить. Решением может стать ввод дополнительной характеристики, помимо лиц.

Обнаружение глаз и деактивация орудия

Продолжая цикл `for`, перебирающий изображения коридора, код листинга 9.3 использует еще один встроенный в OpenCV каскадный классификатор, который будет искать в списке обнаруженных лиц глаза. Поиск глаз добавит дополнительный шаг верификации, снизив тем самым вероятность возникновения ложноположительных срабатываний. А так как у мутантов глаз нет, то при обнаружении хотя бы одного глаза можно будет предположить, что на изображении человек, и отключить огневую систему турели, чтобы тот мог пройти.

Листинг 9.3. Обнаружение глаз в прямоугольниках лиц и отключение орудия *sentry.py*, часть 3

```
        print(f"Searching {image} for eyes.")
        for rect in face_rect_list:
            for (x, y, w, h) in rect:
              ❶ rect_4_eyes = img_gray[y:y+h, x:x+w]
                eyes = eye_cascade.detectMultiScale(image=rect_4_eyes,
                                                    scaleFactor=1.05,
                                                    minNeighbors=2)
              ❷ for (xe, ye, we, he) in eyes:
                    print("Eyes detected.")
                    center = (int(xe + 0.5 * we), int(ye + 0.5 * he))
                    radius = int((we + he) / 3)
                    cv.circle(rect_4_eyes, center, radius, 255, 2)
                    cv.rectangle(img_gray, (x, y), (x+w, y+h), (255, 255, 255), 2)
                  ❸ discharge_weapon = False
                    break
```

Выводим имя просматриваемого изображения и начинаем перебирать прямоугольники в `face_rect_list`. Если прямоугольник присутствует, перебираем кортеж его координат. Используя эти координаты, создаем из изображения подмассив, в котором будем искать глаза ❶.

Далее вызываем для созданного подмассива каскадный классификатор. Поскольку теперь поиск осуществляется в гораздо меньшей области, аргумент `minNeighbors` можно уменьшить.

Аналогично каскадным классификаторам лиц каскадный классификатор глаз возвращает координаты прямоугольника. Начинаем цикл по этим координатам, именуя их с символом `e` — eye — в конце, чтобы отличать от координат прямоугольников лиц ❷.

Теперь рисуем круг вокруг первого найденного глаза. Это нужно только для визуального подтверждения. Алгоритм же в этот момент уже знает, что глаз найден. Далее вычисляем центр прямоугольника, а затем значение радиуса, который немного больше глаза. С помощью метода OpenCV `circle()` рисуем белый круг вокруг подмассива `rect_4_eyes`.

Теперь рисуем прямоугольник вокруг лица, вызывая метод OpenCV `rectangle()` и передавая ему массив `img_gray`. Показываем изображение две секунды, после чего закрываем окно. Так как подмассив `rect_4_eyes` является частью `img_gray`, круг появится, даже если мы явно не передаем этот подмассив методу `im_show()` (рис. 9.11).

Опознав человека, мы отключаем орудие ❸ и заканчиваем цикл `for`. Для подтверждения наличия лица достаточно определить всего один глаз, поэтому пора переходить к следующему прямоугольнику.

Рис. 9.11. Прямоугольник лица и круг глаза

Пропускаем вошедшего или открываем огонь

Код листинга 9.4 продолжает цикл `for`, определяя развитие событий в случаях, когда орудие либо деактивируется, либо открывает огонь. В первом случае программа показывает изображение с обнаруженным лицом и воспроизводит звук «all clear». Во втором она показывает изображение и воспроизводит аудиозапись пулеметной стрельбы.

Листинг 9.4. Определение действий, исходя из активации или деактивации орудия

sentry.py, часть 4

```
    if discharge_weapon == False:
        playsound(tone_path, block=False)
        cv.imshow('Detected Faces', img_gray)
        cv.waitKey(2000)
        cv.destroyWindow('Detected Faces')
        time.sleep(5)

    else:
        print(f"No face in {image}. Discharging weapon!")
        cv.putText(img_gray, 'FIRE!', (int(width / 2) - 20, int(height / 2)),
                                       cv.FONT_HERSHEY_PLAIN, 3, 255, 3)
        playsound(gunfire_path, block=False)
```

```
        cv.imshow('Mutant', img_gray)
        cv.waitKey(2000)
        cv.destroyWindow('Mutant')
        time.sleep(3)

engine.stop()
```

Начинаем с условия, проверяющего активность орудия. Переменную discharge_weapon (выстрелить из орудия) мы устанавливаем как True, когда выбираем текущее изображение из каталога corridor_5 (листинг 9.2). Если код предыдущего листинга обнаружил в прямоугольнике лица глаз, то состояние меняется на False.

В случае, когда орудие отключается, показываем изображение с подтверждением обнаружения (как на рис. 9.11) и проигрываем звук. Сначала вызываем playsound, передаем ему строку tone_path и устанавливаем аргумент block как False. Таким образом мы позволяем playsound выполняться одновременно с тем, как OpenCV показывает изображение. Если же установить block=True, то изображение мы не увидим, пока не закончится воспроизведение *следующего* за звуком аудио. Показываем изображение две секунды, после чего закрываем его и приостанавливаем программу на пять секунд, используя time_sleep().

Если discharge_weapon по-прежнему True, выводим в оболочку сообщение о том, что турель стреляет («FIRE!»). Используя метод OpenCV putText(), объявляем об этом в центре изображения, которое показываем следом (рис. 9.12).

Рис. 9.12. Пример окна с мутантом

Теперь воспроизводим аудиозапись стрельбы. Используем `playsound`, передавая ему строку `gunfire_path` и устанавливая аргумент `block` как `False`. Обратите внимание, что у вас есть возможность удалить строки `root_dir` и `gunfire_path` в листинге 9.1, если вы предоставите полный путь при вызове `playsound`. К примеру, на своей машине с Windows я использовал бы следующий:

```
playsound('C:/Python372/book/mutants/gunfire.wav', block=False)
```

Показываем окно две секунды и закрываем. Приостанавливаем программу на три секунды, делая паузу между отображением мутанта и переходом к следующему изображению из каталога corridor_5. По завершении цикла останавливаем движок `pyttsx3`.

Результаты

Наша программа sentry.py компенсировала повреждения турели и позволила ей функционировать без транспондеров. Предварительно она была настроена на сохранение человеческой жизни, но в сложившихся обстоятельствах это могло привести к ужасным последствиям: если мутант войдет в коридор вслед за человеком, то ему удастся миновать защиту (рис. 9.13).

Рис. 9.13. Худший сценарий

Появление в коридоре мутантов также может спровоцировать срабатывание огнестрельного механизма, если люди, которые находятся в коридоре, в этот момент отвернутся от камеры (рис. 9.14).

Рис. 9.14. И это не мог нормально сделать!

Я пересмотрел достаточно ужастиков и научно-фантастических фильмов и понимаю, что в реальных сценариях запрограммировал бы пушку стрелять во все, что движется. К счастью, с подобной этической дилеммой мне, надеюсь, никогда не придется столкнуться.

Обнаружение лиц в видеопотоке

Можно также обнаруживать лица в режиме реального времени, используя видеокамеры. Делается это просто, поэтому обойдемся без отдельного проекта. Введите код из листинга 9.5 либо используйте его цифровую версию video_face_detect.py из каталога Chapter_9, доступного для скачивания с сайта книги. Вам нужно использовать камеру своего компьютера или внешнюю камеру, подключенную к нему.

Листинг 9.5. Обнаружение лиц в видеопотоке

video_face_detect.py

```
   import cv2 as cv

   path = "C:/Python372/Lib/site-packages/cv2/data/"
   face_cascade = cv.CascadeClassifier(path + 'haarcascade_frontalface_alt.xml')

❶  cap = cv.VideoCapture(0)

   while True:
       _, frame = cap.read()
       face_rects = face_cascade.detectMultiScale(frame, scaleFactor=1.2,
                                                  minNeighbors=3)

       for (x, y, w, h) in face_rects:
           cv.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

       cv.imshow('frame', frame)
     ❷ if cv.waitKey(1) & 0xFF == ord('q'):
           break

   cap.release()
   cv.destroyAllWindows()
```

После импорта OpenCV настраиваем путь на каскадный классификатор Хаара, как делали это в блоке ❸ листинга 9.1. Я здесь использую файл haarcascade_frontalface_alt.xml, поскольку он обладает более высокой точностью (меньше ложноположительных результатов), чем файл haarcascade_frontalface_alt.xml, который мы использовали в предыдущем проекте. Далее инстанцируем объект класса `VideoCapture` по имени `cap` (от capture). Передаем этому конструктору индекс видеоустройства, которое планируется использовать ❶. Если у вас всего одна камера, например встроенная в ноутбук, то ее индекс должен быть `0`.

Чтобы удерживать активными и камеру, и процесс обнаружения, используем цикл `while`. Этот цикл захватывает каждый видеокадр и анализирует его на наличие лиц, так же как в случае со статическими изображениями в предыдущем проекте. Алгоритм обнаружения лиц достаточно быстр, чтобы поспеть за непрерывным потоком, несмотря на объем работы, которую ему необходимо проделать.

Для загрузки кадров вызываем метод `read()` объекта `cap`. Он возвращает кортеж, состоящий из булева кода возврата и `NumPy`-объекта `ndarray`, представляющего текущий кадр. Код возврата используется, чтобы проверить, закончились ли кадры при считывании из файла. Но так как здесь мы считываем не из файла, то присваиваем его нижнему подчеркиванию, указывая на незначительность этой переменной.

Далее повторно используем код предыдущего проекта, который находит прямоугольники лиц и рисует их в кадре. Отображаем кадр с помощью метода OpenCV `imshow()`. В случае обнаружения лица программа должна рисовать в этом кадре прямоугольник.

Завершаем цикл нажатием клавиши `Q` ❷, чтобы выйти. Начинаем с вызова метода OpenCV `waitKey()`, которому передаем значение, равное 1 мс. Этот метод приостанавливает программу при ожидании нажатия клавиши, но ненадолго, чтобы не прерывать видеопоток.

Встроенная в Python функция `ord()` получает в качестве аргумента строку и возвращает переданный аргумент в форме кодовой точки Юникода, в данном случае *q* нижнего регистра. Таблицу сопоставления символов с числами можете найти на http://www.asciitable.com/. Чтобы этот код выполнялся в любой операционной системе, включите побитовый оператор `AND`, `&`, с шестнадцатеричным числом FF(0xFF), которое представляет значение 255. Использование `& 0xFF` гарантирует считывание только 8 последних бит переменной.

Когда цикл завершается, вызываем метод `release()` объекта `cap`. Это освобождает камеру для других приложений. Завершаем программу, закрывая окно отображения.

Можно добавить в процесс обнаружения лиц дополнительные каскады, чтобы повысить точность, как мы делали в предыдущем проекте. Если это слишком замедлит обнаружение, попробуйте уменьшить масштаб видеокадра. Сразу после вызова `cap.read()` добавьте следующий фрагмент:

```
frame = cv.resize(frame, None, fx=0.5, fy=0.5,
                  interpolation=cv.INTER_AREA)
```

Аргументы `fx` и `fy` определяют масштабирование для измерений *x* и *y* экрана. Использование значения `0.5` приведет к уменьшению изначального размера окна вдвое.

Программа будет хорошо отслеживать ваше лицо, но только до тех пор, пока вы не станете совершать нестандартные действия, например не наклоните голову набок. Этого вполне достаточно, чтобы сбить с толку алгоритм обнаружения, из-за чего прямоугольник исчезнет (рис. 9.15).

Каскадные классификаторы Хаара спроектированы для распознавания вертикально расположенных лиц в фас или в профиль, и делают это неплохо. Они способны обрабатывать даже очки и бороду, но стоит лишь наклонить голову, и алгоритм может не справиться.

Рис. 9.15. Обнаружение лиц с использованием видеокадров

Неэффективный, но простой способ распознавания наклоненной головы — использование цикла, который слегка поворачивает изображения, прежде чем передавать их для распознавания лиц.

Небольшой наклон каскадные классификаторы Хаара обработать способны (рис. 9.16), поэтому перед каждой передачей изображение можно поворачивать примерно на 5 градусов, что даст неплохой шанс на получение положительного результата.

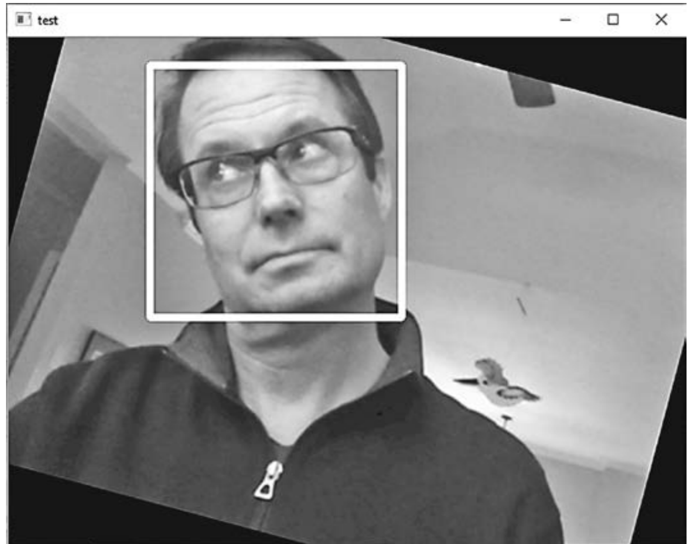


Рис. 9.16. Поворот изображения помогает обнаружению

Использование для обнаружения лиц признаков Хаара — весьма популярный подход, потому что он достаточно быстро выполняется в реальном времени, требуя небольшого количества ресурсов. Однако вы не ошибетесь, если предположите, что существуют и более точные, сложные и ресурсоемкие техники.

К примеру, OpenCV содержит точный и надежный опознаватель лиц, основанный на фреймворке глубокого обучения Caffe. Если хотите узнать об этом детекторе подробнее, почитайте руководство «Face Detection with OpenCV and Deep Learning» на https://www.pyimagesearch.com/.

Еще один вариант — использовать предлагаемый OpenCV каскадный классификатор LBP (Local Binary Patterns), также предназначенный для обнаружения лиц. Эта техника делит изображение лица на блоки, а затем извлекает из них гистограммы локальных бинарных шаблонов (LBPH, Local Binary Pattern Histograms). Подобный способ подтвердил свою эффективность при обнаружении лиц *в естественных условиях* — то есть лиц, которые плохо выровнены и имеют схожие положения на изображениях. Я познакомлю вас с LBPH в следующей главе, где речь пойдет уже о *распознавании* лиц, а не о простом их обнаружении.

Итоги

В этой главе вы научились работать с предоставляемым OpenCV каскадным классификатором Хаара для обнаружения человеческих лиц, библиотекой `playsound` для проигрывания аудиофайлов и `pyttsx3` для воспроизведения аудио по принципу синтеза речи по тексту. Благодаря всем этим полезным инструментам мы смогли быстро написать программу для обнаружения лиц, которая воспроизводила голосовые предупреждения и инструкции.

Дополнительная литература

«Rapid Object Detection Using a Boosted Cascade of Simple Features», или метод Виолы — Джонса (Conference on Computer Vision and Pattern Recognition, 2001), — это разработанный Полом Виолой (Paul Viola) и Майклом Джонсом (Michel Jones) фреймворк для обнаружения объектов, который первым предоставил приличную для практического применения скорость их обработки в реальном времени. Именно он положен в основу техники обнаружения лиц, использованной в текущей главе.

Сайт Адриана Роузброка (Adrian Rosebrock) https://www.pyimagesearch.com/ — прекрасный ресурс для создания механизмов поиска по фотографиям, где вы также найдете много интересных проектов из области компьютерного зрения, например программы, которые обнаруживают огонь и дым, находят цели

в видеопотоках дронов, отличают живые лица от их статичных изображений, автоматически распознают номерные знаки и делают многое другое.

Практический проект: размытие лиц

Вы наверняка видели документальный сюжет или новости, где лицо человека размыто с целью сохранения анонимности, как показано на рис. 9.17. Что ж, этот крутой эффект несложно реализовать с помощью OpenCV. Нужно просто извлечь прямоугольник лица из рамки, размыть его и записать обратно поверх изображения изначальной рамки, при желании добавив прямоугольник, очерчивающий лицо.

Рис. 9.17. Пример размытия лица с помощью OpenCV

Размытие усредняет пиксели в локальной матрице, называемой *ядром*. Можно представить ядро как область внутри рамки на изображении. Все пиксели в этой рамке усредняются до одного значения. Чем больше рамка, тем больше пикселей подвергаются этой операции и тем более гладким становится изображение. По сути, можно рассматривать размытие как фильтр низких частот, который блокирует высокочастотные характеристики, такие как резкие края.

Размытие — единственное в этом процессе, чем мы еще не занимались. Для его реализации используем метод `blur()`, передав ему изображение и кортеж размера ядра в пикселях.

```
blurred_image = cv.blur(image, (20, 20))
```

В этом примере мы замещаем значение заданного пикселя в `image` на среднее значение всех пикселей в квадрате 20 × 20, где этот пиксель — центральный. Данная операция повторяется для каждого пикселя `image`.

Решение под названием practice_blur.py вы найдете в приложении, а также в каталоге Chapter_9, доступном для скачивания с сайта книги.

Усложняем проект: обнаружение кошачьих мордочек

Как оказалось, на планете LV-666 обитают три формы жизни: люди, мутанты и кошки. Счастливый талисман команды — мисс Китти — свободно гуляет по базе и вполне может оказаться в коридоре 5.

Отредактируйте и откалибруйте sentry.py так, чтобы мисс Китти могла спокойно пройти. Это непросто, поскольку кошки не привыкли следовать приказам. Чтобы заставить ее хотя бы взглянуть в камеру, вам может потребоваться добавить в `pyttsx3` команды «Here Kitty, Kitty» («Китти, Китти, сюда») или «Puss, puss, puss» («ксс, ксс, ксс»). А еще лучше применить звук открывания банки тунца, используя `playsound`.

Классификаторы Хаара для кошачьих мордочек можете найти в том же каталоге OpenCV, что и классификаторы, которые мы использовали в проекте 13, а изображение пустого коридора empty_corridor.png — в доступном для скачивания каталоге Chapter_9. Выберите несколько изображений кошек из интернета или своей личной коллекции и вставьте их в разные участки пустого коридора. Чтобы задать для изображения кошки правильный размер, используйте для масштабирования фигуры людей.

10

Ограничение доступа по принципу распознавания лиц

В предыдущей главе вы попробовали себя в качестве техника в составе пехоты Звездных сил коалиции. Здесь ваша роль останется прежней, а вот задача будет уже посложнее. Теперь надо не просто обнаруживать лица, а распознавать их. Ваш командующий, капитан Демминг, обнаружил лабораторию с межпространственным порталом, рождающим монстров, и хочет, чтобы доступ туда был только у него.

Как и в предыдущей главе, следует действовать быстро, поэтому доверимся Python и OpenCV. В частности, мы воспользуемся алгоритмом построения гистограммы на основе локальных бинарных шаблонов (LBPH). Это один из самых старых и простых в применении алгоритмов распознавания лиц, с его помощью мы и заблокируем доступ в лабораторию. Если вы еще не установили OpenCV, то обратитесь к разделу «Установка библиотек Python» на с. 31.

Распознавание лиц с помощью LBPH

При распознавании лиц алгоритм LBPH использует векторы признаков. Как говорилось в главе 5, вектор признаков, по сути, представляет список чисел в определенном порядке. В случае LBPH эти числа отражают некоторые характеристики лица. Для распознавания лиц требуется небольшое число измерений,

например расстояние между глаз, ширина рта, длина носа и ширина лица. Эти четыре параметра, данные по порядку в сантиметрах, могут сформировать такой вектор признаков `(5.1, 7.4, 5.3, 11.8)`. Представление лиц в базе данных с помощью этих векторов ускоряет поиск, а также позволяет выражать отличие между векторами в численной величине, или *расстоянии*.

Конечно же, компьютерное распознавание лиц требует наличия более четырех признаков. При этом разные алгоритмы работают с разными признаками. Сейчас доступны Eigenfaces, LBPH, Fisherfaces, масштабно-инвариантная трансформация признаков (SIFT, scale-invariant feature transform), устойчивые ускоренные признаки (SURF, speeded up robust features) и различные подходы с применением искусственных нейронных сетей. Когда изображения лиц делаются в контролируемых условиях, эти алгоритмы работают с высокой точностью, практически как люди.

К подобным контролируемым условиям при получении изображений лиц могут относиться фронтальный вид каждого лица с нормальным расслабленным выражением и для возможности использования всеми алгоритмами единообразное освещение и разрешение. Лицо не должно быть закрыто волосами, а глаза — очками, если алгоритм обучался распознавать лица именно в таких условиях.

Схема распознавания лиц

Прежде чем перейти к подробностям алгоритма LBPH, рассмотрим общий принцип распознавания лиц. Он состоит из трех основных этапов: захвата, обучения и прогнозирования.

На этапе захвата мы собираем изображения, которые будем использовать для обучения распознавателя лиц (рис. 10.1). Для каждого лица, которое нужно распознавать, требуется не менее десятка изображений с разными выражениями.

Следующий этап захвата — обнаружить лицо на изображении, обвести его прямоугольником, вырезать содержимое прямоугольника, изменить размер вырезанных изображений до одинаковых (размер определяется алгоритмом) и преобразовать их в оттенки серого. Как правило, алгоритмы отслеживают лица при помощи целых чисел, поэтому каждому субъекту потребуется уникальный ID. После обработки изображения сохраняются в одном каталоге, который мы назовем базой данных.

Следующий этап — обучение алгоритма распознавания лиц (рис. 10.2). Алгоритм — в нашем случае LBPH — анализирует каждое из обучающих изображений и записывает результаты в файл YAML (.yml). YAML — это язык сериализации данных в удобочитаемый формат, который изначально расшифровывался как «Yet Another Markup Language» («Еще один язык разметки»), но теперь означает

«YAML Ain't Markup Language» («YAML не является языком разметки»). Таким образом подчеркивается, что он больше чем просто инструмент разметки документов.

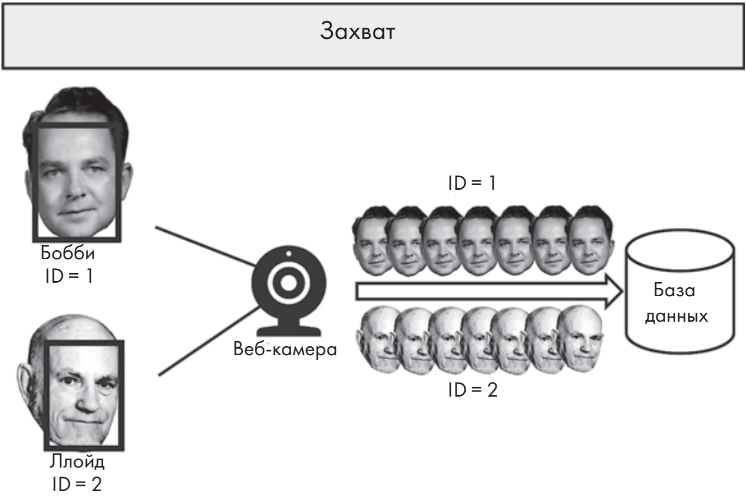


Рис. 10.1. Захват изображений лица для обучения распознавателя лиц

Рис. 10.2. Обучение алгоритма распознавания и запись результатов в файл

Завершающий этап обучения алгоритма распознавания — загрузка нового, неизвестного изображения лица и прогнозирования, кому оно принадлежит (рис. 10.3). Эти изображения подготавливают аналогично изображениям для обучения — то есть их обрезают, подгоняют в размер и преобразуют в оттенки серого. Затем распознаватель их анализирует, сравнивает результаты с лицами в файле YAML и прогнозирует наилучшее совпадение.

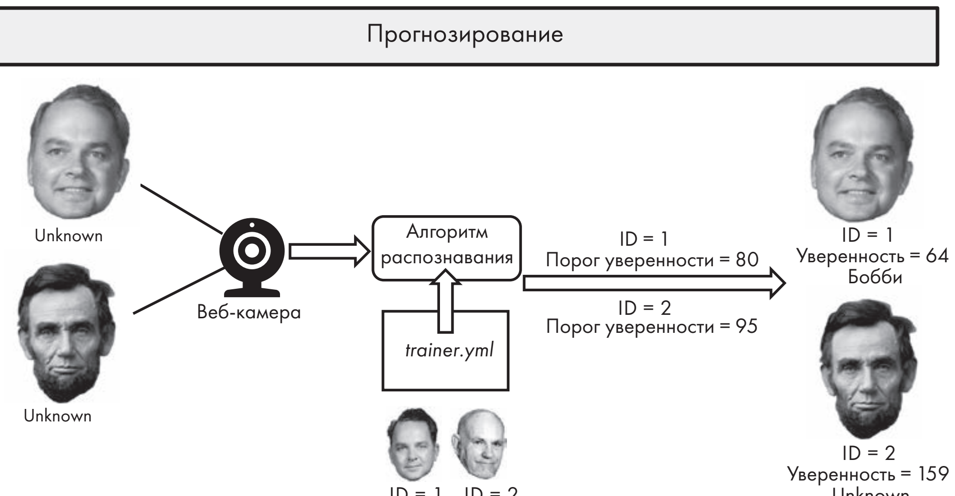


Рис. 10.3. Прогнозирование неизвестных лиц при помощи обученного алгоритма распознавания

Обратите внимание, что алгоритм распознавания осуществляет прогноз личности каждого лица. Если в файле YAML хранится всего один вариант, то алгоритм распознавания присваивает каждому лицу ID «обученного» или, лучше сказать, контрольного лица. Он также выводит значение *уверенности*, представляющее измерение расстояния между новым и контрольным лицами. Чем больше это значение, тем хуже совпадение. Чуть позже мы рассмотрим это подробнее, а сейчас просто знайте, что для принятия решения о верности прогноза лица используется пороговое значение. Если уровень уверенности выше установленного порога, программа отменяет совпадение и определяет проверяемое лицо как unknown (рис. 10.3).

Извлечение гистограмм локальных бинарных шаблонов

Предлагаемый OpenCV алгоритм распознавания лиц основан на локальных бинарных шаблонах (LBP). Эти дескрипторы текстур впервые были применены в 1994 году для описания и классификации текстур поверхностей, различения между бетоном и ковровым покрытием и пр. Лица также состоят из текстур, значит, эта техника вполне годится для их распознавания.

Прежде чем извлекать гистограммы, нужно сгенерировать бинарные шаблоны. Алгоритм LBP вычисляет локальное представление текстуры, сравнивая каждый пиксель с соседними. Первый шаг вычисления — смещение небольшого окна по изображению лица и запись информации о пикселях. На рис. 10.4 показан пример такого окна.

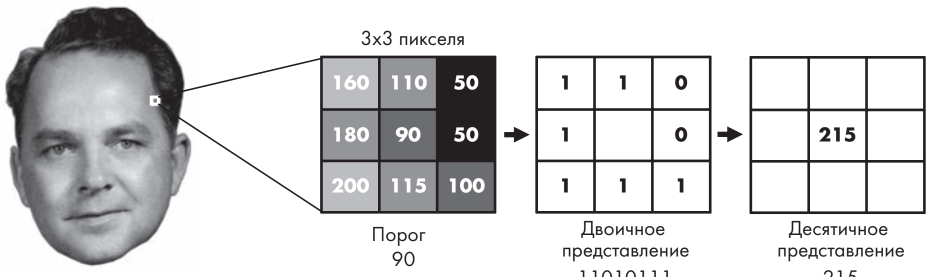


Рис. 10.4. Образец скользящего окна 3 × 3 пикселя, используемого для записи локальных бинарных шаблонов

Следующий шаг — конвертация пикселей в двоичный формат с использованием центрального значения (в данном случае 90) в качестве порога. Для этого сравниваются восемь соседних значений с указанным порогом. Если соседнее значение оказывается равным или больше него, ему присваивается 1. Если же оно меньше порога, присваивается 0. Далее, игнорируя центральное значение, конкатенируем двоичные значения строка за строкой (некоторые методы это делают по часовой стрелке), формируя новое двоичное значение (11010111). Завершаем процесс конвертацией этого двоичного числа в десятичное (215) и сохранением его в позиции центрального пикселя.

Продолжаем смещение окна до тех пор, пока все пиксели не будут конвертированы в значения LBP. Помимо использования для записи соседних пикселей квадратного окна, алгоритм может использовать значение радиуса, реализуя процесс под названием *круговой LBP*.

Теперь пора извлекать гистограммы из LBP-изображения, созданного ранее. Для этого мы используем сетку, чтобы разделить LBP-изображение на прямоугольные области (рис. 10.5). В каждой области построим гистограмму из значений LBP (на рис. 10.5 они называются гистограммой локальной области).

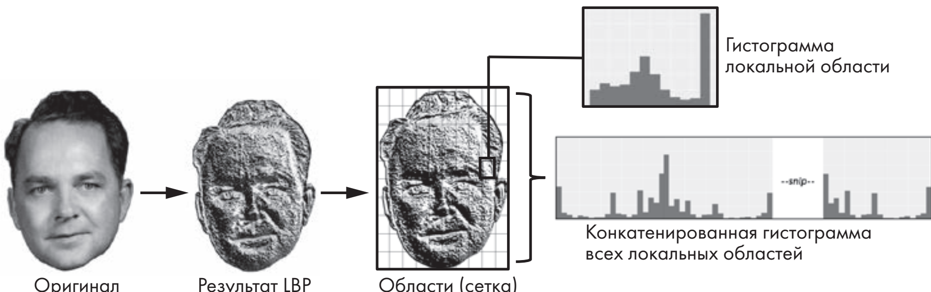


Рис. 10.5. Извлечение гистограмм LBP

После построения гистограмм локальных областей нормализуем и конкатенируем их в одну длинную гистограмму (в урезанном виде показана на рис. 10.5). Поскольку мы используем изображение в оттенках серого с показателем интенсивности от 0 до 255, каждая гистограмма содержит 256 значений. Если вы применили сетку 10 × 10, как на рис. 10.5, то в финальной гистограмме получится 10 × 10 × 256 = 25 600 состояний. Предполагается, что эта составная гистограмма включает признаки, необходимые для распознавания лица. Таким образом, эти признаки являются *представлениями* изображения лица, и процесс распознавания состоит из сравнения представлений, а не самих изображений.

Для прогнозирования личности нового, неизвестного лица, мы извлекаем его конкатенированную гистограмму и сравниваем ее с гистограммами из базы данных. Сравнение — это измерение расстояния между гистограммами. Для вычисления могут использоваться различные методы, включая евклидову метрику, абсолютное расстояние, хи-квадрат и т. д. Алгоритм возвращает ID контрольного изображения с наиболее близким совпадением гистограммы и показатель уверенности. После можно применить к этому показателю порог, как показано на рис. 10.3. Если уверенность относительно оцениваемого изображения окажется ниже порогового значения, значит, совпадение найдено.

Так как OpenCV инкапсулирует все эти шаги, реализация алгоритма LBPH сложностей не вызывает. Этот алгоритм также дает отличные результаты в контролируемой среде и не подвержен влиянию изменения освещения (рис. 10.6).

Алгоритм LBPH отлично реагирует на изменения в освещении, так как опирается на сравнение значений интенсивности пикселей. Даже если освещение в одном изображении оказывается намного ярче, чем в другом, относительная отражательная способность лица остается прежней и LBPH способен ее определить.

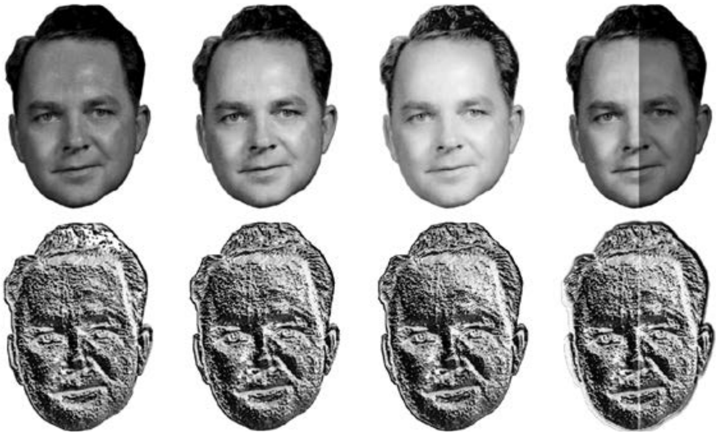


Рис. 10.6. Алгоритм LBP устойчив к изменению освещения

Проект #14. Ограничение доступа к инопланетному артефакту

Ваш взвод прорвался в лабораторию, где находится портал, рождающий монстров. Капитан Демминг приказал немедленно заблокировать эту лабораторию, оставив доступ только для него одного. Один из техников изменит текущие параметры системы с помощью военного ноутбука. В результате капитан Демминг получит доступ через этот ноутбук, используя два уровня безопасности: ввод пароля и верификацию лица. Зная о ваших навыках работы с OpenCV, он приказал вам реализовать распознавание лица.

ЗАДАЧА

Написать программу на Python, которая распознает лицо капитана Демминга.

Стратегия

Вы ограничены во времени и работаете во враждебном окружении, так что нужно использовать быстрый и простой инструмент с хорошими рекомендациями, например предлагаемый OpenCV алгоритм распознавания лиц LBPH. Нам известно, что LBPH лучше всего работает в контролируемых условиях, поэтому для захвата обучающих изображений и лиц любого, кто попытается войти в лабораторию, мы используем одну камеру ноутбука.

Помимо изображений лица Демминга, нужны также изображения лиц других людей. Они позволят нам убедиться в том, что все совпадения действительно относятся к лицу капитана. Можете не беспокоиться о настройке пароля, изолировании программы от пользователя или взломе текущей системы. Всем этим займется второй техник, пока вы будете мочить мутантов.

Поддержка модулей и файлов

Большую часть работы в этом проекте мы проделаем с помощью OpenCV и `NumPy`. Если вы еще не установили эти библиотеки, обратитесь к разделу «Установка библиотек Python» на с. 31. Вам также понадобятся пакеты `playsound` для проигрывания звуков и `pyttsx3` для реализации функциональности синтеза речи по тексту. Подробнее об этих модулях, включая инструкции по их установке, можете почитать в разделе «Код» на с. 269.

Код и сопутствующие файлы находятся в каталоге Chapter_10 на сайте книги https://nostarch.com/real-world-python/. Не меняйте структуру каталогов и имена

файлов после их загрузки (рис. 10.7). Обратите внимание, что каталоги tester и trainer будут созданы позже, в скачиваемых материалах вы их не найдете.

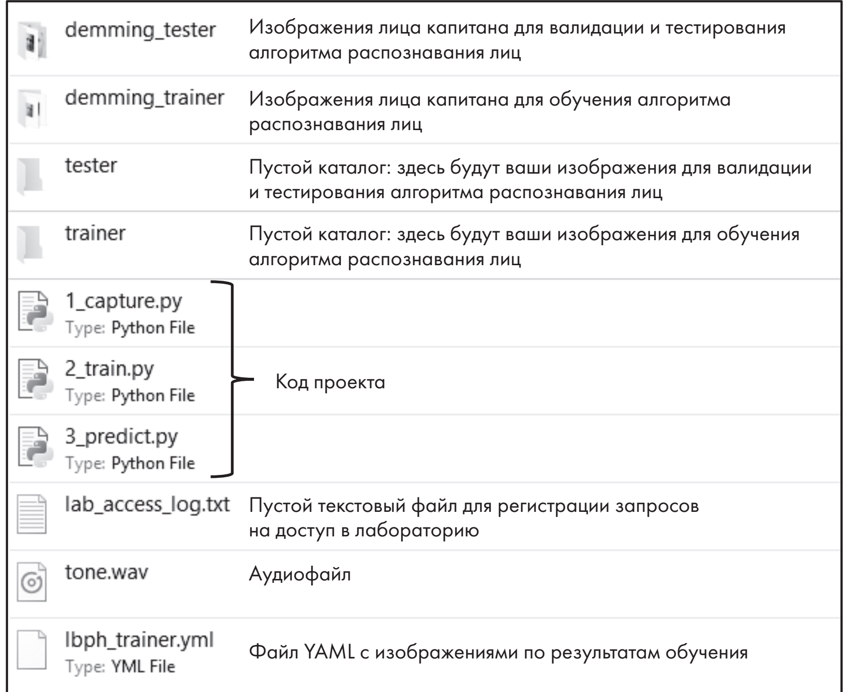


Рис. 10.7. Файловая структура для проекта 14

Каталоги demming_trainer и demming_tester содержат изображения лица капитана Демминга и других людей, которые можно использовать для этого проекта. В текущем виде код ссылается именно на эти каталоги.

Если же вы решите предоставить собственные изображения — например, вы захотите попробовать себя в роли капитана Демминга, — то используйте каталоги с названиями trainer и tester. Первый будет создан автоматически приведенным ниже кодом, а вот каталог tester придется создать самостоятельно, после чего добавить в него собственные фотографии — об этом далее. Естественно, также потребуется скорректировать код, чтобы он ссылался на эти каталоги.

Код для захвата видео

Первый шаг (выполняемый кодом из 1_capture.py) — захват изображений лиц, которые потребуются для обучения алгоритма распознавания. Его можно

пропустить, если вы планируете использовать изображения из каталога demming_trainer.

Если вы хотите попробовать себя в роли капитана Демминга, сделайте через веб-камеру ПК с десяток снимков своего лица с разным выражением и без очков. Если веб-камеры у вас нет, пропустите этот шаг и сделайте несколько селфи с помощью телефона, после чего сохраните их в каталог trainer, как показано на рис. 10.7.

Импорт модулей, настройка аудио, веб-камеры, инструкций и путей файлов

Код листинга 10.2 импортирует модули, инициализирует и настраивает аудиодвижок, а также каскадный классификатор, инициализирует камеру и предоставляет инструкции для пользователя. Каскадный классификатор нам потребуется, так как прежде, чем распознать лицо, его необходимо обнаружить. Чтобы вспомнить об этих классификаторах и обнаружении лиц, обратитесь к разделу «Обнаружение лиц на фотографиях» на с. 264.

Листинг 10.1. Импорт модулей, указание файлов аудио и детектора лиц, а также настройка камеры и инструкций

1_capture.py, часть 1

```
   import os
   import pyttsx3
   import cv2 as cv
   from playsound import playsound

   engine = pyttsx3.init()
❶  engine.setProperty('rate', 145)
   engine.setProperty('volume', 1.0)

   root_dir = os.path.abspath('.')
   tone_path = os.path.join(root_dir, 'tone.wav')

❷  path = "C:/Python372/Lib/site-packages/cv2/data/"
   face_detector = cv.CascadeClassifier(path +
                                        'haarcascade_frontalface_default.xml')

   cap = cv.VideoCapture(0)
   if not cap.isOpened():
       print("Could not open video device.")
❸  cap.set(3, 640) # Frame width.
   cap.set(4, 480) # Frame height.

   engine.say("Enter your information when prompted on screen. \
               Then remove glasses and look directly at webcam. \
               Make multiple faces including normal, happy, sad, sleepy. \
               Continue until you hear the tone.")
   engine.runAndWait()
```

```
❹ name = input("\nEnter last name: ")
  user_id = input("Enter assigned ID Number: ")
  print("\nCapturing face. Look at the camera now!")
```
[1]

Выполняем импорт, как и при обнаружении лиц в предыдущей главе. Операционную систему (через модуль os) мы будем использовать для управления путями файлов, pyttsx3 для воспроизведения аудиоинструкций, cv для работы с изображениями, запуска алгоритма обнаружения лиц и их распознавания, а playsound для проигрывания звука, оповещающего о завершении захвата изображения программой.

Далее настраиваем движок синтеза речи по тексту. С его помощью мы сообщаем пользователю, как выполнять программу. Голос по умолчанию уже определен в вашей операционной системе. В Windows на данный момент это голос Дэвида с американским акцентом ❶, его задает параметр rate. Если вы захотите изменить его, то загляните в инструкции в листинге 9.1 на с. 270.

С помощью звукового сигнала мы будем уведомлять пользователя о завершении процесса захвата видео. Настройте путь к аудиофайлу tone.wav, как в главе 9.

Теперь укажите путь к файлу каскадного классификатора ❷ и присвойте этот классификатор переменной face_detector. Путь, указанный здесь, взят из моего компьютера с Windows, так что ваш может отличаться. К примеру, в macOS эти файлы находятся в opencv/data/haarcascades. Также вы найдете их онлайн по адресу https://hithub.com/opencv/tree/master/fata/haarcascades/.

В главе 9 вы узнали, как делать захват лиц с помощью веб-камеры компьютера. В этой программе мы будем использовать аналогичный код, начиная с вызова cv.VideoCapture(0). Аргумент 0 относится к активной камере. Если у вас их несколько, то, возможно, потребуется опытным путем подобрать другое значение, например 1. С помощью условия проверяем, запустилась ли камера. Если да, то устанавливаем ширину и высоту рамки ❸. Первый аргумент в обоих методах относится к параметру ширины или высоты в списке аргументов.

Из соображений безопасности мы будем контролировать фазу видеозахвата. Для этого используем движок pyttsx3, чтобы объяснить эту процедуру пользователю (таким образом, запоминать ее не придется). Чтобы распознавание было точным, при съемке лица нужно снять очки или другие закрывающие лицо предметы.

[1] Перевод текста аудио: «Введите информацию, когда появится запрос на экране. Затем снимите очки и смотрите прямо в веб-камеру. Сделайте несколько фото лица, включая нейтральное, счастливое, грустное, сонное. Продолжайте, пока не услышите звуковой сигнал». Перевод текста на экране: «Введите свою фамилию: Введите присвоенный ID-номер: Съемка лица. Посмотрите на камеру!»

После этого отснять разные выражения лица. В том числе и такое, которое вы будете принимать при каждом входе в лабораторию.

В завершение вам, как капитану Деммингу, придется выполнять ряд инструкций, выведенных на экране. Прежде всего ввести свою фамилию ❹. В данном случае о повторах беспокоиться не нужно, так как капитан Демминг будет единственным пользователем. Кроме того, мы присвоим ему уникальный номер ID. OpenCV будет использовать эту переменную `user_id`, чтобы отследить все лица в ходе обучения и прогнозирования. Позже мы создадим словарь для сопоставления ID пользователей с конкретными людьми, предполагая, что в будущем доступ в лабораторию получит не только капитан Демминг.

Как только капитан введет свой ID и нажмет `enter`, камера активируется и начнет выполнять захват изображений, о чем нужно сообщить входящему через еще один вызов `print()`. Вспомните, в предыдущей главе я говорил, что каскадный детектор лиц чувствителен к ориентации головы. Для его корректной работы пользователь должен смотреть точно в камеру и держать голову максимально прямо.

Захват изображений для обучения

Листинг 10.2 выполняет захват указанного количества изображений лиц с помощью веб-камеры и цикла `while`. Код сохраняет полученные изображения в каталог, после чего подает сигнал завершения операции.

Листинг 10.2. Захват изображений видео с помощью цикла

1_capture.py, часть 2

```
if not os.path.isdir('trainer'):
os.mkdir('trainer')
    os.chdir('trainer')

frame_count = 0

while True:
    # Поочередный захват 30 кадров.
    _, frame = cap.read()
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
  ❶ face_rects = face_detector.detectMultiScale(gray, scaleFactor=1.2,
                                               minNeighbors=5)
    for (x, y, w, h) in face_rects:
        frame_count += 1
        cv.imwrite(str(name) + '.' + str(user_id) + '.'
                   + str(frame_count) + '.jpg', gray[y:y+h, x:x+w])
        cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv.imshow('image', frame)
        cv.waitKey(400)
  ❷ if frame_count >= 30:
        break
```

```
print("\nImage collection complete. Exiting...")
playsound(tone_path, block=False)
cap.release()
cv.destroyAllWindows()
```

Проверим наличие каталога trainer. Если он отсутствует, создадим его с помощью метода `mkdir()` модуля операционной системы. Далее меняем текущий рабочий каталог на trainer.

Теперь инициализируем переменную `frame_count` как `0`. Код будет захватывать и сохранять видеокадр только при обнаружении в нем лица. Чтобы определить момент завершения программы, нужно отслеживать количество захваченных кадров.

Далее запускаем цикл `while`, установленный как `True`. После вызываем метод `read()` объекта `cap`. Как говорилось в предыдущей главе, этот метод возвращает кортеж из логического кода возврата и объекта `numpy ndarray`, представляющего текущий кадр. Код возврата обычно используется для проверки окончания кадров при считывании из файла. Поскольку здесь считывание мы не делаем, то добавляем к названию файла нижнее подчеркивание, что означает неиспользуемую переменную.

И обнаружение, и распознавание лиц работает с изображениями в оттенках серого, поэтому конвертируем кадр в эту цветовую схему и называем получившийся массив `gray`. Далее вызываем метод `detectMultiscale()` для обнаружения лиц на изображении ❶. Принцип работы этого метода вы найдете в описании листинга 9.2 на с. 273. Поскольку мы контролируем условия получения снимка, заставляя пользователя посмотреть в веб-камеру ноутбука, то в верности работы алгоритма можно не сомневаться, хотя проверить результаты все же нужно.

Предыдущий метод должен вывести координаты очерчивающей лицо прямоугольной рамки. Выполняем цикл `for` для каждого набора координат и тут же увеличиваем переменную `frame_count` на `1`.

С помощью метода OpenCV `imwrite()` сохраняем изображение в каталоге trainer. Каталоги именуются согласно следующей логике: name.user.id.frame_count.jpg — например, demming.1.9.jpg. Сохраняем только часть изображения, попадающую в прямоугольник, очерчивающий лицо, чтобы в обучающий алгоритм не попали лишние фоновые признаки.

Следующие две строки рисуют вокруг лица на оригинальном кадре рамку и показывают ее. Это позволяет пользователю — капитану Деммингу — убедиться, что голова расположена ровно и выражение лица правильное. Метод `waitKey()` откладывает процесс захвата на время, позволяя пользователю «погримасничать», подбирая нужное выражение.

Даже если лицо капитана Демминга при идентификации будет всегда расслабленным, с нейтральным выражением, использование различных выражений при обучении программы позволит добиться более устойчивых результатов. Кстати, весьма полезно, если пользователь в фазе захвата *слегка* покачает головой из стороны в сторону.

Далее проверяем, достигнуто ли заданное количество кадров; при положительном результате цикл завершается ❷. Обратите внимание, если пользователь в камеру не смотрит, то цикл будет выполняться бесконечно, так как он отсчитывает кадры, только если каскадный классификатор обнаруживает лицо и возвращает обрамляющую его рамку.

Сообщаем пользователю о выключении камеры выводом сообщения и звуковым сигналом. Затем завершаем программу, освобождая камеру и закрывая все окна изображений.

На данном этапе в каталоге trainer должно находиться 30 изображений лица пользователя. В следующем разделе, используя эти изображения или изображения из каталога demming_trainer, мы с помощью OpenCV займемся обучением алгоритма распознавания лиц.

Код для обучения алгоритма распознавания лиц

Используем OpenCV для создания алгоритма распознавания лиц на основе LBPH, его обучения на подготовленных изображениях и сохранения результатов в файле. При использовании вашего лица укажите каталог trainer. В ином случае будет использован каталог demming_trainer, который вместе с содержащим код файлом 2_train.py находится в доступной для скачивания директории Chapter_10.

Код листинга 10.3 настраивает пути к каскадам Хаара, используемым для обнаружения лиц, и к обучающим изображениям, захваченным предыдущей программой. OpenCV отслеживает лица с помощью целочисленных меток, а не строк имен. Код также инициализирует списки для хранения этих меток и связанных с ними изображений. Затем выполняются перебор обучающих изображений, их загрузка, извлечение ID пользователя из имени файла и обнаружение лиц. В завершение происходит обучение алгоритма распознавания и сохранение результата в файл.

Листинг 10.3. Обучение и сохранение алгоритма распознавания лиц LBPH

2_train.py

```
import os
import numpy as np
import cv2 as cv

cascade_path = "C:/Python372/Lib/site-packages/cv2/data/"
```

```
face_detector = cv.CascadeClassifier(cascade_path +
                                     'haarcascade_frontalface_default.xml')

❶ train_path = './demming_trainer' # Используйте для лица Демминга
#train_path = './trainer' # Раскомментируйте для использования своего лица
image_paths = [os.path.join(train_path, f) for f in os.listdir(train_path)]
images, labels = [], []

for image in image_paths:
    train_image = cv.imread(image, cv.IMREAD_GRAYSCALE)
  ❷ label = int(os.path.split(image)[-1].split('.')[1])
    name = os.path.split(image)[-1].split('.')[0]
    frame_num = os.path.split(image)[-1].split('.')[2]
  ❸ faces = face_detector.detectMultiScale(train_image)
    for (x, y, w, h) in faces:
    images.append(train_image[y:y + h, x:x + w])
    labels.append(label)
    print(f"Preparing training images for {name}.{label}.{frame_num}")
    cv.imshow("Training Image", train_image[y:y + h, x:x + w])
    cv.waitKey(50)

cv.destroyAllWindows()

❹ recognizer = cv.face.LBPHFaceRecognizer_create()
recognizer.train(images, np.array(labels))
recognizer.write('lbph_trainer.yml')
print("Training complete. Exiting...")
```

Код, выполняющий импорт и распознавание лиц, вы уже видели. В программе 1_capture.py мы вырезали из обучающих изображений рамки с лицами, но нелишне повторить эту процедуру. Поскольку 2_train.py является автономной программой, лучше ничего не принимать без подтверждения.

Далее необходимо выбрать набор обучающих изображений: либо свои из каталога trainer, либо заранее подготовленные из каталога demming_trainer ❶. Закомментируйте или удалите строку для тех, которые не используете. Напомню, так как мы не предоставляем полный путь к каталогу, то нужно запускать программу из папки, где она хранится. Она должна располагаться на один уровень выше каталогов trainer и demming_trainer.

Далее с помощью спискового включения создаем список `image_paths`. В нем мы сохраним путь и имена файлов всех изображений в обучающем каталоге. После создаем пустой список для изображений и их меток.

Начинаем перебирать пути к изображениям циклом `for`. Считываем изображение в оттенках серого, после чего извлекаем из его имени файла числовую метку и преобразуем ее в целое число ❷. Напомню, что метка соответствует ID пользователя, введенному через программу 1_capture.py перед захватом видеокадров.

Давайте подробнее остановимся на процессе извлечения и конвертации. Метод `os.path.split()` получает путь к каталогу и возвращает кортеж из этого пути и имени файла, как показано в следующем фрагменте:

```
>>> import os
>>> path = 'C:\demming_trainer\demming.1.5.jpg'
>>> os.path.split(path)
('C:\\demming_trainer', 'demming.1.5.jpg')
```

Затем, используя индекс `-1`, мы выбираем последний элемент кортежа и разделяем его по точкам. В итоге получается список из четырех элементов (имя пользователя, его ID, номер кадра и расширение файла).

```
>>> os.path.split(path)[-1].split('.')
['demming', '1', '5', 'jpg']
```

Чтобы извлечь значение метки, мы выбираем в этом списке второй элемент с помощью индекса `1`.

```
>>> os.path.split(path)[-1].split('.')[1]
'1'
```

Повторяем этот процесс для извлечения `name` и `frame_num` для каждого изображения. В данный момент все они представлены строками, поэтому ID пользователя нужно преобразовать в целое число, чтобы можно было использовать его в качестве метки.

Теперь вызываем детектор лиц для каждого обучающего изображения ❸. В ответ мы получим `numpy.ndarray`, который назовем `faces`. Начинаем перебор этого массива, содержащего координаты обнаруженных рамок с лицами. Добавляем изображение в рамке в созданный ранее список `images`, а также ID пользователя с этого изображения в список `labels`.

Сообщаем пользователю о происходящем, выводя сообщение в оболочку. Затем в качестве проверки показываем каждое обучающее изображение в течение 50 мс. Если вы смотрели популярный клип Питера Гэбриэла «Sledgehammer» 1986 года, то должны оценить подобный видеоряд.

Теперь пора обучать алгоритм распознавания лиц. Как и в случае с детектором лиц OpenCV, сначала мы инстанцируем объект распознавателя ❹. Далее вызываем метод `train()`, которому передаются списки `images` и `labels`, преобразуемые в массив `NumPy`.

Мы не хотим заниматься обучением алгоритма распознавания при каждой проверке чьего-либо лица, поэтому записываем результаты обучения в файл lbph_trainer.yml и сообщаем пользователю о завершении программы.

Код для прогнозирования лиц

Время переходить к распознаванию лиц. Этот процесс называется *прогнозированием* ввиду того, что в нем все сводится к вероятности. Программа 3_predict.py сначала вычисляет конкатенированную гистограмму для каждого лица, после чего находит расстояние между этой гистограммой и всеми гистограммами в обучающем наборе. Затем она присваивает новому лицу метку и имя контрольного лица с наибольшим соответствием, но только если расстояние между ними впишется в заданный пороговый диапазон.

Импорт модулей и подготовка алгоритма распознавания лиц

Код листинга 10.4 импортирует модули, подготавливает словарь для хранения ID и имен пользователей, настраивает детектор и алгоритм распознавания лиц, а также определяет путь к тестовым данным. Тестовые данные состоят из изображений лица капитана Демминга и нескольких других людей. Изображение капитана из обучающего каталога включено с целью проверки результатов. Если все сработает верно, то алгоритм должен положительно определить это изображение с низким показателем расстояния.

Листинг 10.4. Импорт модулей и подготовка к обнаружению и распознаванию лиц

3_predict.py, часть 1

```
   import os
   from datetime import datetime
   import cv2 as cv

   names = {1: "Demming"}
   cascade_path = "C:/Python372/Lib/site-packages/cv2/data/"
   face_detector = cv.CascadeClassifier(cascade_path +
                                      'haarcascade_frontalface_default.xml')

❶  recognizer = cv.face.LBPHFaceRecognizer_create()
   recognizer.read('lbph_trainer.yml')

   #test_path = './tester'
❷  test_path = './demming_tester'
   image_paths = [os.path.join(test_path, f) for f in os.listdir(test_path)]
```

После нескольких уже знакомых нам операций импорта мы создаем словарь для связывания ID пользователей с их именами. Несмотря на то что сейчас в нем будет всего одна запись, этот словарь `name` в дальнейшем позволит без проблем добавить и другие. Если вы используете собственное лицо, то фамилию можете тоже поменять, но в качестве ID необходимо оставить `1`.

Далее повторяем код, настраивающий объект `face_detector`. При этом введите собственный `cascade_path` (см. листинг 10.1 на с. 298).

Теперь создаем объект распознавателя, как делали это в коде 2_train.py ❶. Затем с помощью метода `read()` загружаем файл .yml, содержащий информацию для обучения.

Алгоритм распознавания нужно протестировать при помощи изображений лиц из отдельного каталога. Если вы используете заранее подготовленные снимки лица капитана Демминга, то укажите путь к каталогу demming_tester ❷. В ином случае укажите путь к созданному ранее каталогу tester, куда можете добавить собственные изображения. Если вы выбрали свой снимок, то не нужно повторно использовать здесь обучающие изображения, хотя одно можно взять в качестве проверочного. Используйте программу 1_capture.py для создания новых изображений. Если вы носите очки, включите ряд фото в очках и без них. При этом также следует добавить сюда несколько снимков из каталога demming_tester.

Распознавание лиц и обновление журнала доступа

Код листинга 10.5 перебирает изображения в тестовом каталоге, обнаруживает на них лица, сравнивает гистограммы этих лиц с содержащимися в файле обучения, именует лицо, присваивает ему значение уверенности, а затем регистрирует его имя и время доступа в текстовом файле. При положительном совпадении ID программа теоретически должна разблокировать вход в лабораторию, но так как лаборатории у нас нет, эту часть мы пропустим.

Листинг 10.5. Выполнение распознавания лиц и обновление журнала доступа

3_predict.py, часть 2

```
for image in image_paths:
    predict_image = cv.imread(image, cv.IMREAD_GRAYSCALE)
    faces = face_detector.detectMultiScale(predict_image,
                                           scaleFactor=1.05,
                                           minNeighbors=5)
    for (x, y, w, h) in faces:
        print(f"\nAccess requested at {datetime.now()}.")
      ❶ face = cv.resize(predict_image[y:y + h, x:x + w], (100, 100))
        predicted_id, dist = recognizer.predict(face)
      ❷ if predicted_id == 1 and dist <= 95:
            name = names[predicted_id]
            print("{} identified as {} with distance={}"
                  .format(image, name, round(dist, 1)))
          ❸ print(f"Access granted to {name} at {datetime.now()}.",
                  file=open('lab_access_log.txt', 'a'))
        else:
            name = 'unknown'
            print(f"{image} is {name}.")

        cv.rectangle(predict_image, (x, y), (x + w, y + h), 255, 2)
        cv.putText(predict_image, name, (x + 1, y + h - 5),
```

```
            cv.FONT_HERSHEY_SIMPLEX, 0.5, 255, 1)
    cv.imshow('ID', predict_image)
    cv.waitKey(2000)
    cv.destroyAllWindows()
```

Начинаем с перебора изображений в тестовом каталоге — в demming_tester либо в tester. Считываем каждое изображение в оттенках серого и присваиваем получающийся массив переменной `predict_image`, после чего выполняем для него обнаружение лиц.

Теперь перебираем рамки лиц, как делали это ранее. Выводим сообщение о запросе доступа, а затем с помощью OpenCV изменяем размер подмассива `face` до 100 × 100 пикселей ❶. Это значение близко к размеру обучающих изображений из каталога demming_trainer. Синхронизировать размер изображений не обязательно, но это улучшит результаты. Если вы используете собственные изображения в качестве изображений капитана Демминга, то следует убедиться, что размеры обучающего и тестового изображений близки.

Теперь пора прогнозировать личность того, чье лицо идентифицируется. Для этого потребуется всего одна строка. В ней мы просто вызываем для объекта `recognizer` метод `predict()`, передавая ему подмассив `face`. Этот метод вернет ID и значение расстояния.

Чем меньше значение расстояния, тем выше вероятность того, что лицо спрогнозировано верно. Это значение можно использовать в качестве порога: все изображения, спрогнозированные как капитан Демминг и имеющие показатель, равный или ниже порога, будут определены как капитан Демминг. Все остальные получат обозначение `'unknown'`.

Порог задаем с помощью инструкции `if` ❷. Если вы используете собственные обучающие и тестовые изображения, то при первом запуске программы установите значение расстояния как 1000. Просмотрите эти значения для всех изображений в тестовом каталоге, включая известные и `'unknown'`. Найдите пороговое значение, ниже которого все лица верно распознаются как принадлежащие капитану Деммингу. В дальнейшем это будет ваш дискриминатор. Для изображений в каталогах demming_trainer и demming_tester пороговое расстояние должно равняться 95.

Далее получаем имя для изображения, используя в качестве ключа в словаре `names` значение `predicted_id`. Выводим в оболочке сообщение о распознавании изображения, включая имя соответствующего файла, имя из словаря и значение расстояния.

Для журнала выводим сообщение о том, что `name` (в данном случае капитан Демминг) получил доступ в лабораторию, и с помощью модуля `datetime` указываем время доступа ❸.

Не забудьте о необходимости фиксировать данные о входе и выходе людей из лаборатории. Вот один удобный прием для этого: просто напишите файл с помощью функции `print()`. Откройте файл lab_access_log.txt и включите в него параметр `a`, означающий «добавить». Таким образом, вместо переписывания файла для каждого нового изображения вы будете добавлять в его конец новую строку. Вот пример содержимого этого файла:

```
Access granted to Demming at 2020-01-20 09:31:17.415802.
Access granted to Demming at 2020-01-20 09:31:19.556307.
Access granted to Demming at 2020-01-20 09:31:21.644038.
Access granted to Demming at 2020-01-20 09:31:23.691760.
--snip--
```

Если условие не выполняется, устанавливаем `name` как `'unknown'` и выводим соответствующее сообщение. Затем рисуем рамку вокруг лица и с помощью метода OpenCV `putText()` указываем имя пользователя. Показываем изображение в течение двух секунд, после чего уничтожаем его.

Результаты

Ниже на рис. 10.8 показаны результаты выбора из 20 изображений в каталоге demming_tester снимков капитана Демминга. Прогнозирующий код верно определил снимки капитана Демминга без ложноположительных результатов.

Рис. 10.8. Демминги и не Демминги

Для того чтобы алгоритм LBPH работал максимально точно, нужно использовать его в контролируемых условиях. Напомню, что, вынудив пользователя получать доступ через ноутбук, мы контролировали его позу, размер лица, разрешение изображения и освещение.

Итоги

В этой главе мы реализовали распознавание лиц людей с помощью алгоритма построения гистограмм локальных бинарных шаблонов. Используя всего несколько строк кода, мы создали надежный алгоритм распознавания лиц, который способен с легкостью обрабатывать изменчивые условия освещения. Кроме того, с помощью метода `os.path.split()` из стандартной библиотеки мы разделили пути к каталогам и имена файлов для создания настраиваемых имен.

Дополнительная литература

В «Local Binary Patterns Applied to Face Detection and Recognition» (Polytechnic University of Catalonia, 2010) автор Лаура Мария Санчес Лопес (Laura María Sánchez López) тщательно разбирает LBPH. В формате PDF эту книгу вы найдете онлайн, например, на сайте https://www.semanticscholar.org/.

В «Look at the LBP Histogram» с сайта AURlabCVsimulator (https://aurlabcvsimulator.readthedocs.io/en/latest/) приведен код на Python, который позволяет визуализировать изображение LBPH.

Если вы являетесь пользователем macOS или Linux, то обязательно ознакомьтесь с библиотекой `face_recognition`, созданной Адамом Гейтгеем (Adam Geitgey). Это простая и очень точная система распознавания лиц, работающая на принципах глубокого обучения. Инструкции по ее установке и описание вы найдете на сайте Python Software Foundation по адресу https://pypi.org/project/face_recognition/.

«Machine Learning Is Fun! Part 4: Modern Face Recognition with Deep Learning» (Medium, 2016) Адама Гейтгея — короткий и интересный обзор современных техник распознавания лиц при помощи Python, OpenFace и `dlib`.

«Liveness Detection with OpenCV» (PyImageSearch, 2019), написанное Адрианом Роузброком (Adrian Rosebrock), — онлайн-руководство о том, как защитить систему распознавания от спуфинговых атак, при которых злоумышленник может обмануть алгоритм, например, поднеся к видеокамере фотографию капитана Демминга.

В различных городах и колледжах в разных странах мира начали запрещать использование систем распознавания лиц. При этом изобретательные граждане

тоже не остаются в стороне, с целью защиты личности они разрабатывают специальную одежду, вводящую системы в заблуждение. «These Clothes Use Outlandish Designs to Trick Facial Recognition Software into Thinking You're Not Human» («В этой одежде используется неординарный дизайн, благодаря которому системы не могут распознать человека») Аарона Холмса (Aaron Holmes) (Business Insider, 2020 год) и «How to Hack Your Face to Dodge the Rise of Facial Recognition Tech» («Как замаскировать лицо, чтобы избежать обнаружения всепроникающими системами распознавания лиц») Элизы Томас (Elise Thomas) (Wired, 2019) — обзор практичных и непрактичных способов решения этой проблемы.

«OpenCV Age Detection with Deep Learning» (PyImageSearch, 2020) Адриана Роузброка — онлайн-руководство по использованию OpenCV для прогнозирования возраста человека по его фотографии.

Усложняем проект: добавление пароля и видеозахвата

Написанная нами в проекте 14 программа 3_predict.py перебирает каталог фотографий, выполняя распознавание лиц. Перепишите ее так, чтобы она динамически распознавала лица в видеопотоке камеры. Рамка лица и имя должны появляться в видеокадре так же, как на изображениях из каталога.

Сделайте так, чтобы для запуска программы пользователю нужно было ввести пароль, который будет проверяться. На случай ввода верного пароля нужно будет также добавить аудиообращение к пользователю с просьбой посмотреть в камеру. Если программа верно распознает капитана Демминга, то использовать еще одну аудиозапись, сообщающую о предоставлении доступа в лабораторию. В противном случае воспроизводить сообщение об отказе в доступе.

Если вам понадобится помощь в распознавании лица в видеопотоке, то загляните в программу challenge_video_recognize.py в приложении к книге. Обратите внимание, что в случае с видеорядом вам может потребоваться использовать более высокое значение уверенности, чем при работе с фотографиями.

Для того чтобы иметь возможность отслеживать попытки доступа в лабораторию, сохраните один кадр в тот же каталог в виде файла lab_access_log.txt. В качестве имени файла используйте зарегистрированные результаты из `datetime.now()`, чтобы можно было сопоставить лицо с попыткой доступа. Обратите внимание, что нужно переформатировать строку, возвращенную из `datetime.now()`, чтобы она содержала только такие символы, которые допускаются вашей операционной системой для имен файлов.

Усложняем проект: похожие лица и близнецы

Используйте код из проекта 14 для сравнения лиц двойников и близнецов знаменитостей. Обучите алгоритм распознавания на фотографиях лиц из интернета и проверьте, удастся ли вам обмануть алгоритм LBPH. Для сравнения можно взять Скарлетт Йохансcон и Эмбер Херд, Эмму Уотсон и Кирнан Шипку, Лиама Хемсворта и Карена Качанова, Роба Лоу и Йэна Сомерхолдера, Хилари Дафф и Викторию Педретти, Брайс Даллас Ховард и Джессику Честейн, а также Уилла Феррелла и Чада Смита.

В качестве известных близнецов — например, астронавтов Марка и Скотта Келли, а также знаменитостей Мэри-Кейт и Эшли Олсен.

Усложняем проект: машина времени

Если вы вдруг решите посмотреть какой-нибудь старый сериал, то увидите в нем известных актеров в молодости или юности. Но даже несмотря на то, что люди хорошо распознают лица, у нас могут возникнуть сомнения при взгляде, например, на молодого Иэна Маккеленна или Патрика Стюарта. Именно поэтому лишь определенная интонация речи или особая манера поведения побуждают нас уточнить у Google, кто играет в данном фильме.

Алгоритмы также могут ошибаться при распознавании состарившихся людей. Чтобы увидеть, как алгоритм LBPH работает в подобных случаях, возьмите программу из проекта 14 и обучите ее на ваших снимках (или ваших родственников), сделанных, например, в ранней юности. А потом протестируйте алгоритм распознавания на снимках, сделанных позже.

11

Создание интерактивной карты побега от зомби

В 2010 году на телевизионном канале AMC был показан сериал «Ходячие мертвецы». События фильма разворачиваются в начале зомби-апокалипсиса, когда небольшая группа людей пытается выжить на территории Атланты, штат Джорджия. Этот сериал получил широкое признание критиков и стал самым просматриваемым за всю историю кабельного телевидения. По его мотивам даже было снято продолжение под названием «Бойтесь ходячих мертвецов», и возник абсолютно новый жанр на телевидении — шоу-обсуждение очередного эпизода, которое в данном случае получило название «Говорящие мертвецы».

В этой главе вы будете играть роль проницательного ученого, который предвидит надвигающийся крах цивилизации. Вам нужно подготовить карту, которая поможет выжившим из сериала «Ходячие мертвецы» перебраться из густонаселенного района Атланты в более пустынную область к западу от Миссисипи. Вам понадобится библиотека `pandas` для загрузки, анализа и очистки данных, а также модули `bokeh` и `holoviews` для рисования карты.

Проект #15. Визуализация плотности населения с помощью хороплетной карты

По данным ученых (да, они это реально изучали), выжить во время зомби-апокалипсиса смогут те, кто обитает в максимально отдаленном от городов месте.

На карте Соединенных Штатов на рис. 11.1 это темные области. Чем ярче огни, тем выше плотность население в этом районе. Так что если мы хотим избегать встреч с людьми, то «нельзя идти на свет».

Рис. 11.1. Освещение разных районов территории США при ночной съемке в 2012 году

К сожалению, выжившие в Атланте находятся очень далеко от относительно безопасного запада США. Им придется миновать множество крупных и небольших городов, в идеале прокладывая путь по наименее населенным областям. Карты, предлагаемые на автозаправках, не содержат информации о населенности регионов, а вот в переписи населения эти данные есть. Так что, прежде чем цивилизация сгинет и интернет отрубится, успейте скачать данные о плотности населения на свой ноутбук, чтобы разобраться с ними позже при помощи Python.

Лучше всего представить этот тип данных через *хороплетную карту*. Она позволяет с помощью цветов и шаблонов визуализировать статистику в заданных географических регионах. Возможно, вам знакомы хороплетные карты по президентским выборам в США: в случае победы республиканцев территории округов окрашиваются в красный цвет, а при победе демократов — в синий (рис. 11.2).

Если у выживших будет подобная карта, показывающая плотность населения (например, число людей на квадратную милю в каждом округе), то они смогут определить самые короткие и, теоретически, безопасные маршруты из Атланты и через Юг Штатов. Несмотря на то что данные переписи населения содержат

еще более точную информацию, нам вполне достаточно данных по округам. Стада зомби из «Ходячих мертвецов» мигрируют, когда становятся голодны, что быстро делает подробную статистику бесполезной.

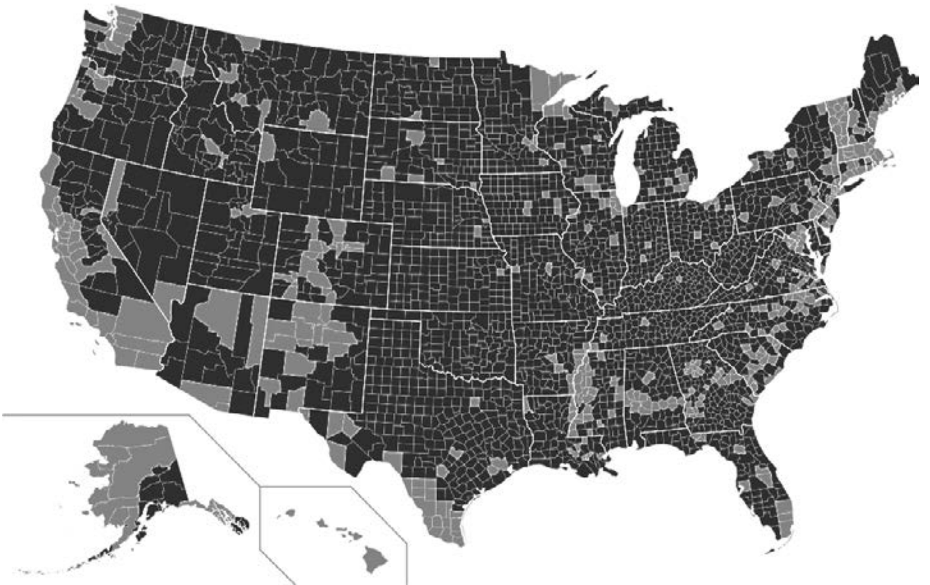


Рис. 11.2. Хороплетная карта результатов президентских выборов в США в 2016 году (светло-серый = демократы, темно-серый = республиканцы)

Для определения оптимальных маршрутов выжившие могут использовать бумажные карты автомобильных дорог штата, которые доступны на автозаправках и в информационно-туристических центрах. Эти карты включают в себя границы округов, что упрощает сопоставление сети указанных в них городов и дорог с распечаткой хороплетной карты страничного размера.

ЗАДАЧА

Создать интерактивную карту 48 смежных штатов, на которой будет отражена плотность населения в каждом округе.

Стратегия

Как и любые задачи визуализации данных, эта решается следующим образом: поиск и очистка данных, выбор типа графического представления и инструмента для отображения данных, их подготовка к отрисовке и итоговое изображение.

В нашем случае найти данные просто, так как информация о переписи населения в США является публичной. Но ее еще нужно *очистить*, выявив и обработав ложные точки данных и нулевые значения и решив проблемы с форматированием. В идеале нужно также убедиться в точности информации. Это сложная задача, которую специалисты по обработке данных, похоже, слишком часто пропускают. По меньшей мере данные следует проверить с позиции здравого смысла, что в принципе может подождать до момента их отрисовки. Например, Нью-Йорк должен иметь большую плотность населения, чем, скажем, Биллингс, Монтана.

Далее следует решить, в каком виде представлять данные. Мы воспользуемся картой, но в качестве альтернативы можно взять гистограмму или таблицу. При этом очень важно выбрать правильный инструмент — в нашем случае библиотеку Python, с помощью которого мы будем создавать графическое представление. Выбор инструмента оказывает существенное влияние на подготовку данных и конечное изображение.

Несколько лет назад одна из сетей быстрого питания демонстрировала рекламный ролик, в котором клиент утверждал, что любит «разнообразие, но не когда его слишком много». В случае с инструментами визуализации в Python можно смело заявить, что вариантов слишком много при не слишком заметном различии между ними: `matplotlib`, `seaborn`, `plotly`, `bokeh`, `folium`, `altair`, `pygal`, `ggplot`, `holoviews`, `cartopy`, `geoplotlib`, а также встроенные в `pandas` функции.

Эти всевозможные библиотеки визуализации имеют свои достоинства и недостатки, но так как наш проект требует скорости, мы возьмем простую в использовании библиотеку `holoviews`, подкрепив ее `bokeh` для рисования графических изображений. Эта комбинация позволит создать интерактивную карту, написав всего несколько строк кода, а `bokeh`, что весьма удобно, содержит схемы полигонов штатов и округов США.

Выбрав инструмент визуализации, приведем данные в подходящий для него формат. Как заполнять получаемые из файла формы округов данными о населении из другого файла? Здесь нам не обойтись без некоторого реверс-инжиниринга с использованием примера кода из галереи `holoviews`. Затем мы отрисуем карту с помощью `bokeh`.

К счастью, данные практически всегда анализируются с помощью библиотеки `pandas`. Этот модуль позволит загрузить информацию о переписи населения и переформатировать ее для использования в `holoviews` и `bokeh`.

Библиотека анализа данных

Открытая библиотека `pandas` является наиболее популярным инструментом для извлечения, обработки и управления данными в Python. Она содержит структуры

данных, спроектированные для работы со стандартными источниками, такими как реляционные базы данных SQL и электронные таблицы Excel. Если вы планируете работать с данными в будущем, то однозначно столкнетесь с `pandas`.

Эта библиотека содержит две основные структуры данных: серии и датафреймы. *Серия* — это одномерный массив, который может содержать любой тип данных, например целые числа, числа с плавающей точкой, строки и т. д. А поскольку в основе `pandas` лежит `NumPy`, то объект серии, по сути, представляет собой два связанных массива (если о массивах вы ничего не знаете, загляните в главу 1, на с. 38). Один массив содержит значения точек данных, которые могут иметь любой тип данных `NumPy`. Второй содержит метки для каждой точки данных, называемые *индексами* (табл. 11.1).

Таблица 11.1. Объект серии

Индекс	Значение
0	25
1	432
2	–112
3	99

В отличие от индексов элементов списка, в Python индексы в серии не обязательно должны быть целыми числами. В табл. 11.2 это имена людей с возрастом в качестве значений.

Таблица 11.2. Объект серии со смысловыми метками

Индекс	Значение
Хавьер	25
Кэрол	32
Лора	19
Сара	29

Аналогично спискам или массивам `NumPy` серии можно разделять или выбирать их отдельные элементы, указывая индекс. Сериями можно управлять по-разному, например фильтровать их, выполнять математические операции, а также объединять их с другими сериями.

Датафрейм более сложен и состоит уже из двух измерений. Это табличная структура со столбцами, строками и данными, как в электронных таблицах

(табл. 11.3). Можно рассматривать его как упорядоченную коллекцию столбцов с двумя индексированными массивами.

Таблица 11.3. Объект датафрейм

Индекс	**Столбцы**			
	Страна	**Штат**	**Округ**	**Население**
0	США	Алабама	Отога	54 571
1	США	Алабама	Болдуин	182 265
2	США	Алабама	Барбур	27 457
3	США	Алабама	Бибб	22 915

Первый индекс, для строк, работает аналогично массиву индексов в сериях. Второй отслеживает серии меток, где каждая метка представляет заголовок столбца. Датафреймы также напоминают словари; имена столбцов — ключи, а серии данных в каждом столбце — значения. Такая структура позволяет с легкостью управлять ими.

Разбор всей функциональности `pandas` занял бы целую книгу, к тому же много соответствующей информации опубликовано онлайн. Так что мы отложим дальнейшее обсуждение этой темы до листинга, где рассмотрим конкретные примеры применения функциональности `pandas`.

Библиотеки bokeh и holoviews

Модуль `bokeh` (https://bokeh.org/) является открытой интерактивной библиотекой визуализации для современных браузеров. Он предлагает элегантные графические решения для объемных или потоковых датасетов. Графические изображения получаются при помощи HTML и JavaScript, которые на сегодня лидируют среди языков программирования, используемых для создания интерактивных веб-страниц.

Открытая библиотека `holoviews` (https://holoviews.org) позволяет упростить анализ и визуализацию данных. Вместо построения графика через прямые вызовы к графической библиотеке, например `bokeh` или `matplotlib`, вы создаете описание данного объекта, и графические изображения становятся автоматическими визуальными представлениями этого объекта.

В галерее примеров `holoviews` есть несколько хороплетных карт, визуализированных с помощью `bokeh` (например, http://holoviews.org/gallery/demos/bokeh/texas_choropleth_example.html). Позже мы разберем пример с визуализацией

данных об уровне безработицы, чтобы понять, как аналогичным образом представить данные о плотности населения.

Установка pandas, bokeh и holoviews

Если вы проработали проект из главы 1, то `pandas` и `NumPy` у вас уже установлены. Если же нет, обратитесь за инструкциями к разделу «Установка библиотек Python» на с. 31.

Один из вариантов установить `holoviews` вместе с последней версией всех рекомендованных пакетов для работы с этим модулем под Linux, Windows или macOS — использовать Anaconda.

```
conda install -c pyviz holoviews bokeh
```

Таким образом вы установите библиотеку графики `matplotlib` и более интерактивную библиотеку графики `bokeh`, служащие в качестве бэкенда[1], а также блокнот Jupyter/IPython.

Аналогичный набор пакетов можно установить с помощью `pip`.

```
pip install 'holoviews[recommended]'
```

Через `pip` также доступны дополнительные минимальные настройки, если `bokeh` у вас уже установлена. Эти и прочие инструкции вы найдете на страницах http://holoviews.org/install.html и http://holoviews.org/user_guide/Installing_and_Configuring.html.

Работа с данными по уровню безработицы и плотности населения в округах и штатах

Библиотека `bokeh` содержит файлы данных о границах округов и информацию об уровне безработицы в США для каждого округа от 2009 года. Как уже говорилось, с помощью данных о безработице мы разберемся, как отформатировать данные о плотности населения, которые возьмем из переписного листа 2010 года.

Чтобы скачать образцы данных `bokeh`, подключитесь к интернету, откройте оболочку Python и введите:

```
>>> import bokeh
>>> import bokeh.sampledata
>>> bokeh.sampledata.download()
```

[1] Фронтендом в данном контексте является пользовательский код, например для отрисовки карты, а бэкенд при этом выполняет всю внутреннюю работу по этой отрисовке. — *Примеч. пер.*

```
Creating C:\Users\lee_v\.bokeh directory
Creating C:\Users\lee_v\.bokeh\data directory
Using data directory: C:\Users\lee_v\.bokeh\data
```

Как видите, программа сообщает вам, куда помещает данные, чтобы bokeh могла автоматически их найти. В вашем случае путь будет другой. Более подробно о скачивании образцов данных — на https://docs.bokeh.org/en/latest/docs/reference/sampledata.html.

Найдите в скачанном каталоге файлы US_Counties.csv и unempoyment109.csv. Эти текстовые файлы находятся в распространенном формате CSV. Каждая строка файла представляет запись с несколькими полями, разделенными точками.

Файл с информацией по безработице наглядно демонстрирует трудности специалистов по обработке данных. Откройте его, и вы увидите, что там отсутствуют имена столбцов, описывающие данные в них (рис. 11.3), хотя о значении большинства полей можно догадаться и так. С этим мы разберемся позже.

	A	B	C	D	E	F	G	H	I
1	CN010010	1	1	Autauga County, AL	2009	23,288	21,025	2,263	9.7
2	PA011000	1	3	Baldwin County, AL	2009	81,706	74,238	7,468	9.1
3	CN010050	1	5	Barbour County, AL	2009	9,703	8,401	1,302	13.4
4	CN010070	1	7	Bibb County, AL	2009	8,475	7,453	1,022	12.1
5	CN010090	1	9	Blount County, AL	2009	25,306	22,789	2,517	9.9
6	CN010110	1	11	Bullock County, AL	2009	3,527	2,948	579	16.4

Рис. 11.3. Первые строки файла unemployment09.csv

Теперь откройте файл с информацией по округам Соединенных Штатов. Вы увидите очень много столбцов, но у них хотя бы есть заголовки (рис. 11.4). Наша задача — соотнести данные о безработице (см. рис. 11.3) с географическими данными (см. рис. 11.4): то же самое мы позже проделаем с информацией о плотности населения.

	A	B	C	D	E	F	G	H	I	J	K	L
1	County Name	State-County	state abbr	State Abbr.	geometry	value	GEO_ID	GEO_ID2	Geographic Name	STATE num	COUNTY num	FIPS formula
2	Autauga	AL-Autauga	al	AL	<Polygon>	126.4	05000US01001	1001	Autauga County, Alabama	1	1	1001
3	Baldwin	AL-Baldwin	al	AL	<Polygon>	486.1	05000US01003	1003	Baldwin County, Alabama	1	3	1003
4	Barbour	AL-Barbour	al	AL	<Polygon>	583.3	05000US01005	1005	Barbour County, Alabama	1	5	1005
5	Bibb	AL-Bibb	al	AL	<Polygon>	569.3	05000US01007	1007	Bibb County, Alabama	1	7	1007
6	Blount	AL-Blount	al	AL	<Polygon>	893	05000US01009	1009	Blount County, Alabama	1	9	1009

Рис. 11.4. Первые строки из файла US_Counties.csv

Сами же данные о плотности населения находятся в файле census_data_popl_2010.csv каталога Chapter_11, доступного для скачивания с сайта книги. В оригинале

этот файл назывался DEC_10_SF1_GCTPH1_US05PR_with.ann.csv — я взял его с сайта American FactFinder. Однако к моменту публикации книги правительство США уже перенесло эти данные на новый сайт (вот их новый адрес: https://www.census.gov/data/what-is-data-census-gov.html[1]).

Заглянув в начало файла с информацией о плотности населения, мы увидим множество столбцов с двумя строками заголовков (рис. 11.5). Нас интересует столбец M под названием *Density per square mile of land area — Population* (Плотность на квадратную милю суши — Население).

	A	B	C	D	E	F
1	GEO.id	GEO.id2	GEO.display-label	GCT_STUB.target-geo-id	GCT_STUB.target-geo-id2	GCT_STUB.display-label
2	Id	Id2	Geography	Target Geo Id	Target Geo Id2	Geographic area
3	0100000US		United States	0100000US		United States
4	0100000US		United States	0400000US01	1	United States - Alabama
5	0100000US		United States	0500000US01001	1001	United States - Alabama -
6	0100000US		United States	0500000US01003	1003	United States - Alabama -

	G	H	I	J	K	L	M	N
1	GCT_STUB.display-label	HD01	HD02	SUBHD0301	SUBHD0302	SUBHD0303	SUBHD0401	SUBHD0402
2	Geographic area	Populat	Housin	Area in squa	Area in squa	Area in squa	Density per	Density per
3	United States	3087455	1317047	3796742.23	264836.79	3531905.43	87.4	37.3
4	Alabama	4779736	2171853	52420.07	1774.74	50645.33	94.4	42.9
5	Autauga County	54571	22135	604.39	9.95	594.44	91.8	37.2
6	Baldwin County	182265	104061	2027.31	437.53	1589.78	114.6	65.5

Рис. 11.5. Первые несколько строк census_data_popl_2010.csv

Теперь у вас есть все библиотеки Python и данные, необходимые для генерации хороплетной карты плотности населения *в теории*. Но прежде чем мы сможем перейти к коду, необходимо понять, как именно связывать данные о населении с географическими данными, чтобы правильно вписать информацию по округу в его контуры.

Разбираем код holoviews

Умение адаптировать уже написанный код к собственным целям — очень ценный навык для специалиста по обработке данных. Здесь может потребоваться навык реверс-инжиниринга (обратной разработки). Поскольку открытое ПО является бесплатным, порой оно плохо документировано и приходится разбираться в коде самостоятельно. Итак, попробуем выполнить реверс-инжиниринг для текущей задачи.

[1] На территории России доступ к данному ресурсу ограничен. — *Примеч. ред.*

В предыдущих главах мы использовали примеры из галереи, предоставляемые открытыми модулями `turtle` и `matplotlib`. Библиотека `holoviews` тоже имеет свою галерею (http://holoviews.org/gallery/index.html), которая включает Texas Choropleth Example, хороплетную карту уровня безработицы в Техасе от 2009 года (рис. 11.6).

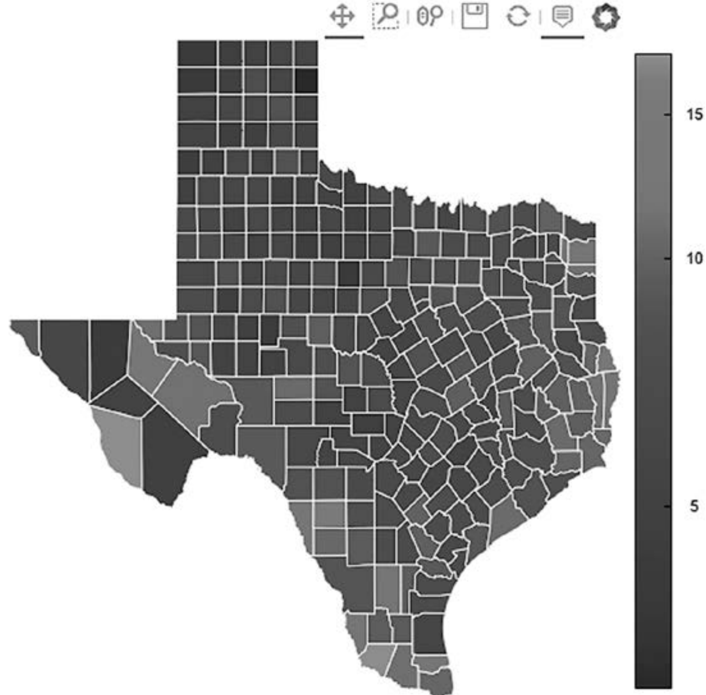


Рис. 11.6. Хороплетная карта уровня безработицы в Техасе от 2009 года из галереи holoviews

Листинг 11.1 содержит код, предоставленный `holoviews` для этой карты. Мы построим на основе данного примера проект, но для этого потребуется разобраться с двумя основными отличиями. Во-первых, мы планируем графически отобразить плотность населения, а не уровень безработицы. Во-вторых, нам нужна карта соседних штатов, а не только Техаса.

Листинг 11.1. Код из галереи holoviews для генерации хороплетной карты Техаса

texas_choropleth_example.html

```
   import holoviews as hv
   from holoviews import opts
   hv.extension('bokeh')
❶  from bokeh.sampledata.us_counties import data as counties
```

```
from bokeh.sampledata.unemployment import data as unemployment

counties = [dict(county, ❷ Unemployment=unemployment[cid])
            for cid, county in counties.items()
          ❸ if county["state"] == "tx"]

choropleth = hv.Polygons(counties, ['lons', 'lats'],
                         [('detailed name', 'County'), 'Unemployment'])

choropleth.opts(opts.Polygons(logz=True,
                              tools=['hover'],
                              xaxis=None, yaxis=None,
                              show_grid=False,
                              show_frame=False,
                              width=500, height=500,
                              color_index='Unemployment',
                              colorbar=True, toolbar='above',
                              line_color='white'))
```

Этот код импортирует набор данных из `bokeh` ❶. Нам нужно узнать формат и содержимое переменных `unemployment` и `counties`. К данным по уровню безработицы мы обратимся позже с помощью переменной `unemployment` и индекса или ключа `cid`, который может означать county ID ❷. Программа выбирает из всех штатов именно Техас на основе условной инструкции, используя код `state` ❸.

Разберем все это в оболочке Python.

```
   >>> from bokeh.sampledata.unemployment import data as unemployment
❶ >>> type(unemployment)
   <class 'dict'>
❷ >>> first_2 = {k: unemployment[k] for k in list(unemployment)[:2]}
   >>> for k in first_2:
           print(f"{k} : {first_2[k]}")
❸ (1, 1) : 9.7
   (1, 3) : 9.1
   >>>
   >>> for k in first_2:
           for item in k:
               print(f"{item}: {type(item)}")
❹ 1: <class 'int'>
   1: <class 'int'>
   1: <class 'int'>
   3: <class 'int'>
```

Начинаем с импорта набора данных из `bokeh`, используя синтаксис из примера галереи. Далее используем встроенную функцию `type()` для проверки типа данных переменной `unemployment` ❶. Вы увидите, что это словарь.

Теперь с помощью функции генерации словаря создаем новый словарь, включающий две первые строки `unemployment` ❷. Выводим результаты и видим, что ключи являются кортежами, а значения — числами, что позволяет отобразить

уровень безработицы в процентах ❸. Проверяем тип данных чисел в ключе. Это целые числа, не строки ❹.

Сравниваем вывод в ❸ с первыми двумя строками файла CSV на рис. 11.3. Первое число в кортеже ключа, вероятно код штата, берется из столбца B. Второе число в кортеже, по всей видимости код округа, берется из столбца C. Показатель уровня безработицы, очевидно, хранится в столбце I.

Теперь сравниваем содержимое `unemployment` с рис. 11.4, где показаны данные округов. *STATE num* (столбец J) и *COUNTY num* (столбец K), очевидно, содержат компоненты кортежа ключа.

Пока все хорошо, но если заглянуть в данные переписи населения на рис. 11.5, то мы не найдем код штата или округа для подстановки в кортеж. Однако есть числа в столбце E, которые совпадают с числами, содержащимися в последнем столбце данных по округам, отмеченном на рис. 11.4 как *FIPS formula*. Похоже, эти числа FIPS относятся к кодам штатов и округов.

На деле оказывается, что код *Федерального стандарта по обработке информации (FIPS)*, по сути, является ZIP-кодом округа. FIPS-код — это код из пяти цифр, присвоенный каждому округу Национальным институтом стандартов и технологий. Первые две его цифры представляют штат округа, а последние три — сам округ (табл. 11.4).

Таблица 11.4. Определение округов США по коду FIPS

Округ США	Код штата	Код округа	FIPS
Округ Болдуин, AL	01	003	1003
Округ Джонсон, IA	19	103	19103

Поздравляю, теперь вы знаете, как сопоставлять данные переписи населения США с контурами округов из набора данных `bokeh`. Пора писать заключительный код!

Код для отрисовки хороплетной карты

Программа choropleth.py включает код и для очистки данных, и для отрисовки хороплетной карты. Копию этого кода вместе с данными о переписи населения вы найдете в каталоге Chapter_11, доступном для скачивания с сайта книги по адресу https://nostarch.com/real-world-python/.

Импорт модулей и данных для построения датафрейма

Код листинга 11.2 импортирует модули и набор данных по округам из `bokeh`, включая координаты для полигонов всех округов США. Он также загружает

и создает объект датафрейма для представления информации о плотности населения. Далее выполняется очистка и подготовка этих данных для совмещения с данными об округах.

Листинг 11.2. Импорт модулей и данных, создание датафрейма и переименование столбцов

choropleth.py, часть 1

```
   from os.path import abspath
   import webbrowser
   import pandas as pd
   import holoviews as hv
   from holoviews import opts
❶  hv.extension('bokeh')
   from bokeh.sampledata.us_counties import data as counties

❷  df = pd.read_csv('census_data_popl_2010.csv', encoding="ISO-8859-1")

   df = pd.DataFrame(df,
                     columns=
                      ['Target Geo Id2',
                     'Geographic area.1',
                     'Density per square mile of land area - Population'])

   df.rename(columns =
             {'Target Geo Id2':'fips',
              'Geographic area.1': 'County',
              'Density per square mile of land area - Population':'Density'},
             inplace = True)

   print(f"\nInitial popl data:\n {df.head()}")
   print(f"Shape of df = {df.shape}\n")
```

Начинаем с импорта `abspath` из библиотеки операционной системы. С ее помощью мы будем искать абсолютный путь к HTML-файлу созданной хороплетной карты. Далее импортируем модуль `webbrowser`, который позволит запустить этот HTML-файл. Данный модуль необходим, так как библиотека `holoviews` спроектирована для работы с блокнотом Jupyter и не сможет автоматически отображать карту без сторонней помощи.

Следом импортируем `pandas` и повторяем импорты `holoviews` из примера с галереей в листинге 11.1. Обратите внимание, что необходимо указать `bokeh` как расширение `holoviews`, или ее *бэкенд* ❶. Дело в том, что `holoviews` может работать с другими графическими библиотеками, например с `matplotlib`, и должна знать, какую именно использовать.

С помощью импорта мы добавили географические данные. Теперь нужно загрузить данные о населении, используя `pandas`. Этот модуль включает набор API-функций ввода-вывода, упрощающих считывание и запись данных. Эти

ридеры (читатели) и *райтеры* (писатели) работают с основными форматами, такими как разделенные запятой значения (read_csv, to_csv), Excel (read_excel, to_excel), язык структурированных запросов (read_sql, to_sql), язык гипертекстовой разметки (read_html, to_html) и другие. В текущем проекте мы будем работать с форматом CSV.

В большинстве случаев можно считывать CSV-файлы, не указывая символьную кодировку.

```
df = pd.read_csv('census_data_popl_2010.csv')
```

Однако в данном случае мы получим ошибку:

```
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf1 in position 31:
invalid continuation byte
```

Дело в том, что файл содержит символы в кодировке Latin-1, также известной как ISO-8859-1, вместо типичной UTF-8. Проблему можно решить, добавив аргумент кодировки ❷.

Теперь через вызов конструктора `DataFrame()` преобразуем файл с данными о плотности населения в табличный датафрейм. Нам не нужны все столбцы из первоначального файла, поэтому передаем в конструктор только имена тех, которые хотим сохранить, — E, G и M на рис. 11.5, то есть код FIPS, название округа (без названия штата) и плотность населения соответственно.

Затем с помощью метода `rename()` укорачиваем метки столбцов и делаем их более смысловыми. Используем имена *fips* (ZIP-код), *County* (округ) и *Density* (плотность населения).

Выводим несколько первых строк датафрейма методом `head()`, а также форму датафрейма, используя его атрибут `shape`. По умолчанию метод `head()` отображает первые пять строк. Если вы хотите увидеть больше, передайте ему нужное число в качестве аргумента, например `head(20)`. В оболочке должен отобразиться следующий вывод:

```
Initial popl data:
      fips         County  Density
0      NaN  United States     87.4
1      1.0        Alabama     94.4
2  1001.0  Autauga County     91.8
3  1003.0  Baldwin County    114.6
4  1005.0  Barbour County     31.0
Shape of df = (3274, 3)
```

Заметьте, что первые две строки (строка 0 и 1) не несут для нас полезной информации. Вообще, становится ясно, что для каждого штата указывается строка

с его названием, которую можно удалить. Атрибут shape показывает, что всего в датафрейме содержится 3274 строки.

Удаление лишних строк с названиями штатов и подготовка кодов штатов и округов

Код листинга 11.3 удаляет все строки, код FIPS которых меньше или равен 100. Это строки заголовков, которые показывают, что здесь начинается информация о следующем штате. После удаления создаются новые столбцы для кодов штатов и округов, которые выводятся из столбца кодов FIPS. Их мы используем позже для выбора подходящих границ округа из образца данных bokeh.

Листинг 11.3. Удаление лишних строк с последующей подготовкой кодов штатов и округов

choropleth.py, часть 2

```
   df = df[df['fips'] > 100]
   print(f"Popl data with non-county rows removed:\n {df.head()}")
   print(f"Shape of df = {df.shape}\n")

❶  df['state_id'] = (df['fips'] // 1000).astype('int64')
   df['cid'] = (df['fips'] % 1000).astype('int64')
   print(f"Popl data with new ID columns:\n {df.head()}")
   print(f"Shape of df = {df.shape}\n")
   print("df info:")
❷  print(df.info())

   print("\nPopl data at row 500:")
❸  print(df.loc[500])
```

Для отображения данных о плотности населения в каждом округе нужно преобразовать их в словарь, где ключи будут кортежами, состоящими из кода штата и кода округа, а значения — данными о плотности населения. Но, как вы уже видели, в информации о населении нет столбцов с кодами штатов и округов; указаны лишь коды FIPS. Так что нам нужно извлечь из этих кодов числа, обозначающие штаты и округа.

Прежде всего, избавляемся от всех строк, не относящихся к округам. Если взглянуть на предыдущий вывод в оболочке (или строки 3 и 4 на рис. 11.5), то мы увидим, что они не содержат четырех- или пятицифровой код FIPS. Значит, можно использовать столбец fips для создания нового датафрейма, также с именем df, который будет хранить только строки со значением fips больше 100. Чтобы убедиться, что все сработало, повторим вывод предыдущего листинга, как показано здесь:

```
 Popl data with non-county rows removed:
      fips          County  Density
2  1001.0  Autauga County     91.8
3  1003.0  Baldwin County    114.6
```

```
4  1005.0  Barbour County     31.0
5  1007.0     Bibb County     36.8
6  1009.0   Blount County     88.9
Shape of df = (3221, 3)
```

Теперь двух ненужных строк в начале датафрейма нет, и атрибут `shape` показывает, что мы избавились от 53 строк. Это заголовки 50 штатов, а также United States, District of Columbia (DC) и Puerto Rico. Обратите внимание, что FIPS-код DC представлен как 11001, а Пуэрто-Рико в дополнение к трехзначному коду округа для своих 78 муниципалитетов использует код штата 72. DC мы оставим, но Пуэрто-Рико позже уберем.

Далее создаем столбцы для значений кодов штатов и округов. Первый назовем `state_id` ❶. Деление на 1000 с округлением вниз (`//`) возвращает частное с удалением цифр после запятой. Поскольку последние три числа кода FIPS зарезервированы для кодов округов, у нас остается код штата.

Несмотря на то что `//` возвращает целое число, по умолчанию в столбце нового датафрейма используются значения с плавающей точкой. Но наш анализ образца данных из `bokeh` показал, что там для этих кодов в представленных кортежами ключах использовались целые числа. Преобразуем столбец в целочисленный тип при помощи метода `astype()` библиотеки `pandas`, передав ему `'int64'`.

Теперь создаем новый столбец для кодов округов. Назовем его `cid`, чтобы он соответствовал терминологии из примера с хороплетной картой `holoviews`. Поскольку нас интересуют три последние цифры кода FIPS, мы используем оператор `modulo` (`%`), получая остаток от деления первого аргумента на второй. Преобразуем этот столбец в целочисленный тип данных, как в предыдущей строке.

Снова делаем вывод, только теперь вызываем для датафрейма метод `info()`❷. Он выводит краткую сводку, включая типы данных и использование памяти.

```
Popl data with new ID columns:
     fips          County  Density  state_id  cid
2  1001.0  Autauga County     91.8         1    1
3  1003.0  Baldwin County    114.6         1    3
4  1005.0  Barbour County     31.0         1    5
5  1007.0     Bibb County     36.8         1    7
6  1009.0   Blount County     88.9         1    9
Shape of df = (3221, 5)
df info:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3221 entries, 2 to 3273
Data columns (total 5 columns):
fips        3221 non-null float64
County      3221 non-null object
Density     3221 non-null float64
state_id    3221 non-null int64
```

```
cid         3221 non-null int64
dtypes: float64(2), int64(2), object(1)
memory usage: 151.0+ KB
None
```

Из столбцов и информационной сводки видно, что значения `state_id` и `cid` представлены целыми числами.

Коды штатов в первых пяти строках состоят из одной цифры, но в других случаях они вполне могут состоять из двух. Просмотрите их значения в остальных строках, вызвав для датафрейма метод `loc()` с переданным в него большим значением номера строки ❸. Это позволит перепроверить коды, состоящие из двух цифр.

```
Popl data at row 500:
fips                13207
County      Monroe County
Density              66.8
state_id               13
cid                   207
Name: 500, dtype: object
```

Столбцы `fips`, `state_id` и `cid` выглядят вполне логично. На этом подготовка данных завершается. Следующий шаг — преобразование этих данных в словарь, который `holoviews` сможет использовать для создания хороплетной карты.

Подготовка к отображению

Листинг 11.4 конвертирует ID штатов и округов, а также данных о плотности населения в отдельные списки. Затем эти списки перестраиваются в словарь того же формата, что и словарь `unemployment`, использованный в примере из галереи `holoviews`. Помимо этого, код листинга перечисляет исключаемые из карты штаты и территории, создавая список из данных, которые нужно отобразить на хороплетной карте.

Листинг 11.4. Подготовка данных о населении к отображению на карте

choropleth.py, часть 3

```
state_ids = df.state_id.tolist()
cids = df.cid.tolist()
den = df.Density.tolist()

tuple_list = tuple(zip(state_ids, cids))
popl_dens_dict = dict(zip(tuple_list, den))

EXCLUDED = ('ak', 'hi', 'pr', 'gu', 'vi', 'mp', 'as')

counties = [dict(county, Density=popl_dens_dict[cid])
            for cid, county in counties.items()
            if county["state"] not in EXCLUDED]
```

Ранее мы рассматривали переменную `unemployment` в примере из галереи `holoviews` и выяснили, что она является словарем. Кортежи из кодов штатов и округов служили ключами, а показатели уровня безработицы — значениями:

```
(1, 1) : 9.7
(1, 3) : 9.1
--snip--
```

Чтобы сформировать аналогичный словарь для данных о населении, мы сначала используем метод `pandas tolist()` для создания отдельных списков столбцов датафрейма `state_id`, `cid` и `Density`. Далее с помощью встроенной функции `zip()` совместим списки кодов штатов и округов в кортежные пары. Итоговый словарь, `popl_dens_dict`, мы создадим, также совмещая полученный `tuple_list` со списком плотности населения (название `tuple_list` не совсем точно; технически это `tuple_tuple`). На этом заключительная подготовка данных завершена.

Выжившим из сериала «Ходячие мертвецы» посчастливится выбраться из Атланты. Но мы не будем предполагать, что они отправятся до Аляски. Создаем кортеж `EXCLUDED`, состоящий из названия штатов и территорий, которые входят в полученные из `bokeh` данные об округах, но не относятся к сопредельным штатам. Речь идет об Аляске, Гавайях, Пуэрто-Рико, Гуаме, Виргинских островах, Северных Марианских островах и Американском Самоа. Чтобы сократить объем ввода, можно использовать сокращения, представленные в отдельном столбце датасета округов (рис. 11.4).

Далее, как и в примере с `holoviews`, создаем словарь и помещаем его в список `counties`. Сюда мы будем добавлять данные о плотности населения. Далее связываем его с соответствующим округом, используя ID округа `cid`. С помощью условной конструкции применяем кортеж `EXCLUDED`.

Если вывести первый индекс списка, получим (обрезанный) вывод:

```
[{'name': 'Autauga', 'detailed name': 'Autauga County, Alabama', 'state':
'al', 'lats': [32.4757, 32.46599, 32.45054, 32.44245, 32.43993, 32.42573,
32.42417, --snip-- -86.41231, -86.41234, -86.4122, -86.41212, -86.41197,
-86.41197, -86.41187], 'Density': 91.8}]
```

Пара «ключ — значение» `Density` теперь замещает пару показателей уровня безработицы из примера галереи `holoviews`. Пора рисовать карту!

Рисуем хороплетную карту

В листинге 11.5 мы создаем хороплетную карту, сохраняем ее в виде файла .html и открываем с помощью `webbrowser`.

Листинг 11.5. Создание и отрисовка хороплетной карты

choropleth.py, часть 4

```
   choropleth = hv.Polygons(counties,
                            ['lons', 'lats'],
                            [('detailed name', 'County'), 'Density'])

❶  choropleth.opts(opts.Polygons(logz=True,
                       tools=['hover'],
                       xaxis=None, yaxis=None,
                       show_grid=False, show_frame=False,
                       width=1100, height=700,
                       colorbar=True, toolbar='above',
                       color_index='Density', cmap='Greys', line_color=None,
                       title='2010 Population Density per Square Mile of
                              Land Area'
                       ))

❷  hv.save(choropleth, 'choropleth.html', backend='bokeh')
   url = abspath('choropleth.html')
   webbrowser.open(url)
```

Согласно документации `holoviews`, класс `Polygons()` создает непрерывную заполненную область в 2D-формате в виде списка полигонов. Создаем переменную `choropleth`, передавая ей переменную `counties` и ключи словаря, включая `lons` (longitude, долгота) и `lats` (latitude, широта), используемые для отрисовки полигонов округов. Также передаем названия округов и ключи данных о плотности населения. Инструмент наведения курсора `holoviews` использует этот кортеж, `('detailed name', 'County')` для показа полного названия округа, например `County: Claiborne County, Mississippi`, при наведении курсора на разные области карты (рис. 11.7).

Далее настраиваем параметры карты ❶. Сначала разрешаем использование логарифмической цветной шкалы путем установки аргумента `logz` как `True`.

Окно `holoviews` по умолчанию содержит набор инструментов, таких как pan (панорамирование), zoom (изменение масштаба), save (сохранение), refresh (обновление) и других (указаны в верхнем правом углу рис. 11.7).

С помощью аргумента `tools` добавляем в этот список функциональность наведения курсора. Это позволит опрашивать карту, получая и название округа, и подробную информацию о плотности населения в нем.

Мы создаем нестандартный график с аннотированными осями *x* и *y*, поэтому устанавливаем их как `None`. Аналогичным образом не отображаем сетку или рамку вокруг карты. Ширину и высоту карты задаем в пикселях. Вам может потребоваться подстроить их значения под свой монитор. Далее устанавливаем `colorbar` (цветовая шкала) как `True` и помещаем `toolbar` (панель инструментов) в верхнюю часть дисплея.

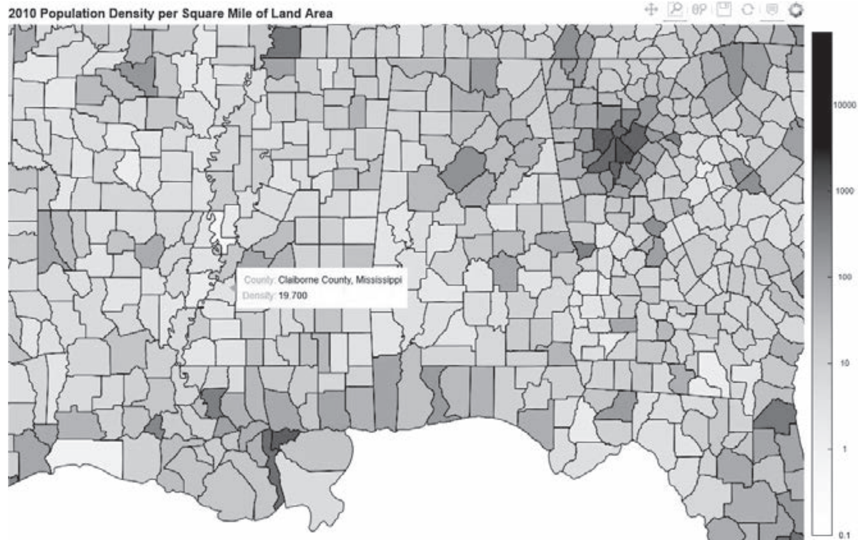


Рис. 11.7. Хороплетная карта с включенной функциональностью курсора

Поскольку мы хотим закрашивать округа согласно плотности их населения, то устанавливаем аргумент `color_index` как `Density`, представляющий значения из `popl_dens_dict`. Для цветов заполнения используем `Greys cmap` (оттенки серого). Если вы решите использовать более яркую палитру, то обратитесь к списку доступных цветовых карт на странице http://build.holoviews.org/user_guide/Colormaps.html. Не забудьте выбрать ту, в чьем имени содержится `bokeh`. Завершаем настройку цветовой схемы выбором цвета линии для контуров округов. Удачными вариантами для серой карты будут `None`, `'white'` или `'black'`.

В конце добавляем название карты. Теперь ее можно отобразить.

Для сохранения карты в текущем каталоге используем метод `holoviews save()`, передавая ему переменную `choropleth`, имя файла с расширением .html и имя использованного бэкенда отрисовки ❷. Как уже говорилось, `holoviews` спроектирована для использования с Jupyter Notebook. Если вам нужно, чтобы карта автоматически всплывала в браузере, сначала присвойте переменной `url` полный путь к ее файлу, после чего этот `url` можно будет открывать с помощью модуля `webbrowser` (рис. 11.8).

С помощью панели инструментов в верхней части карты можно делать панорамирование, масштабирование (используя Box или Lasso), сохранять, обновлять изображение или наводить на его участки указатель. Инструмент наведения, показанный на рис. 11.7, поможет найти менее населенные округа в тех частях карты, где визуально отличить разницу в оттенках сложно.

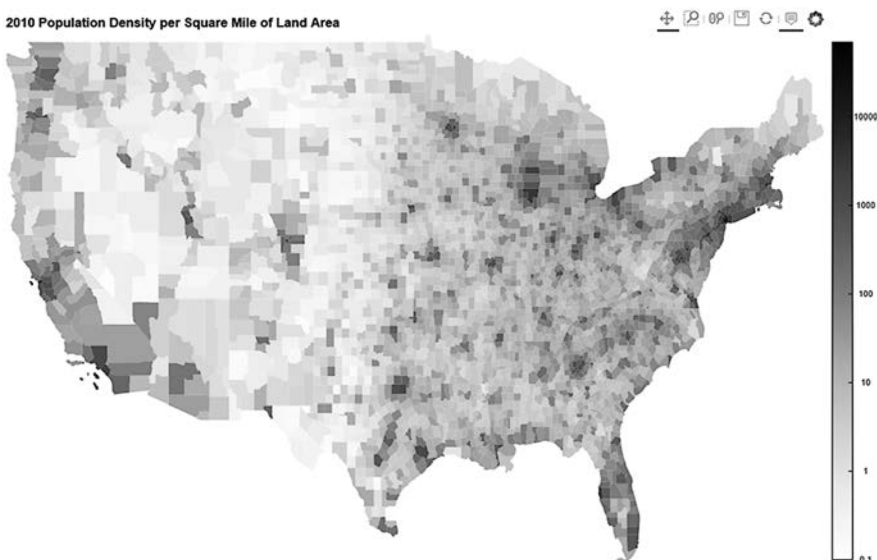


Рис. 11.8. Хороплетная карта плотности населения от 2010 года. Чем светлее область, тем меньше людей здесь проживает

ПРИМЕЧАНИЕ

Инструмент масштабирования Box Zoom делает возможным быстрый просмотр прямоугольной области, но может растягивать или сжимать оси карты. Чтобы сохранить при масштабировании соотношение ее сторон, используйте комбинацию инструментов Wheel Zoom и Pan.

Планирование маршрута

Горы Чисос в Национальном парке Биг-Бенд, где находится потухший супервулкан, могут стать одним из лучших мест на земле для укрытия во время зомби-апокалипсиса. Этот удаленный и похожий на крепость горный массив (рис. 11.9) возвышается над окружающей пустынной местностью, в пике достигая практически 8000 футов (около 2500 м). В середине горного массива располагается природный водный бассейн, а также парковые объекты, включая хижины, домики, магазин и ресторан. Эта область изобилует дикими животными и рыбой, ручьи полны чистой водой, а берега Рио-Гранде вполне подходят для фермерства.

С помощью хороплетной карты можно быстро спланировать маршрут к этой далекой природной крепости. Но для начала нужно безопасно покинуть Атланту. Кратчайший путь из этого района — узкий коридор между городами Бирмингем и Монтгомери в Алабаме (рис. 11.10). Обойти следующий крупный город,

Джексон, штат Миссисипи, можно по его северной либо южной окраине. Но для выбора оптимального пути прежде всего необходимо оценить обстановку.

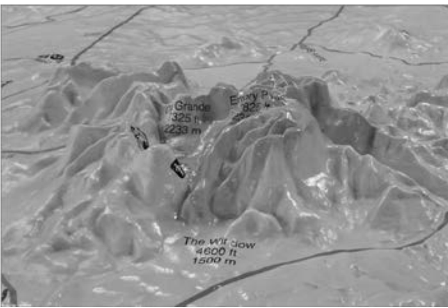


Рис. 11.9. Горы Чисос на Западе Техаса (*слева*) и их отображение на 3D-карте (*справа*)

Рис. 11.10. Побег из Атланты

Южный маршрут вокруг Джексона короче, но вынудит выживших двигаться по коридору между густонаселенными районами I-35, южнее которого располагается Сан-Антонио, а севернее — Даллас-Форт-Уэрт (DFW) (рис. 11.11). Этот потенциально опасный участок в округе Хилл, Техас (на рис. 11.11 отмечен кружочком).

Альтернативный северный маршрут через долину Ред-Ривер, расположенную между Оклахомой и Техасом, длиннее, но безопаснее, особенно если использовать

удобную возможность сплава по реке. Оказавшись на западе от Форт-Уэрта, выжившие смогут пересечь реку и повернуть на юг к земле обетованной.

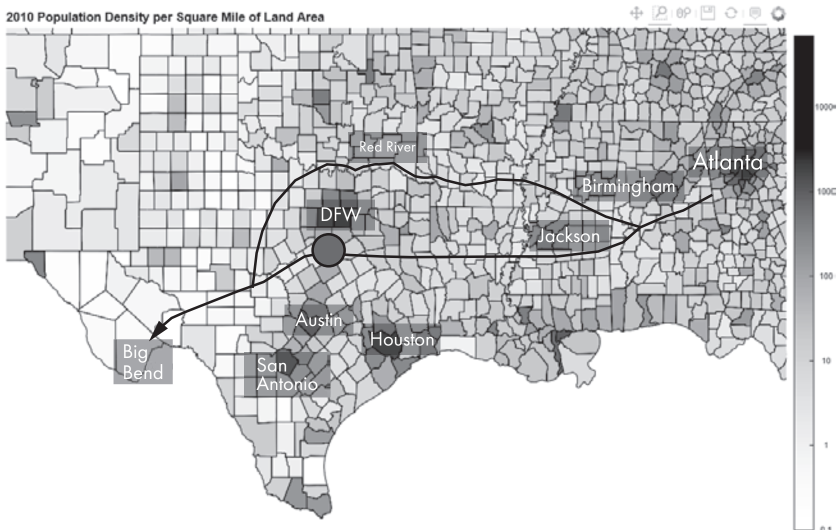


Рис. 11.11. Путь на запад

Планирование оказалось бы еще проще при наличии в `holoviews` инструмента слайдера, позволяющего интерактивно изменять цветовую шкалу. Можно было бы, например, отфильтровывать или изменять тона округов простым перетаскиванием ползунка вверх и вниз по шкале. Это бы упростило поиск маршрутов через наименее населенные округа.

К сожалению, слайдер не входит в список опций окна `holoviews`. Но мы знакомы с `pandas`, так что мы вооружены. Просто добавим следующий фрагмент кода после строки, выводящей информацию в строке 500:

```
df.loc[df.Density >= 65, ['Density']] = 1000
```

Это изменит значения плотности населения в датафрейме, установив для тех, что равны или больше 65, постоянное значение 1000. Еще раз запустите программу, и у вас получится изображение, как на рис. 11.12. С этими новыми значениями опасность путешествия через Сан-Антонио — Остин — Даллас становится более наглядной, как и относительная безопасность долины Ред-Ривер на северной границе восточного Техаса.

Хотите узнать, какой путь выбрали выжившие в ТВ-шоу? Никакой. Первые четыре сезона они провели в окрестностях Атланты, сначала разбив лагерь

в Стоун-Маунтин, а затем отсиживаясь в тюрьме рядом с вымышленным городом Вудбери (рис. 11.13).

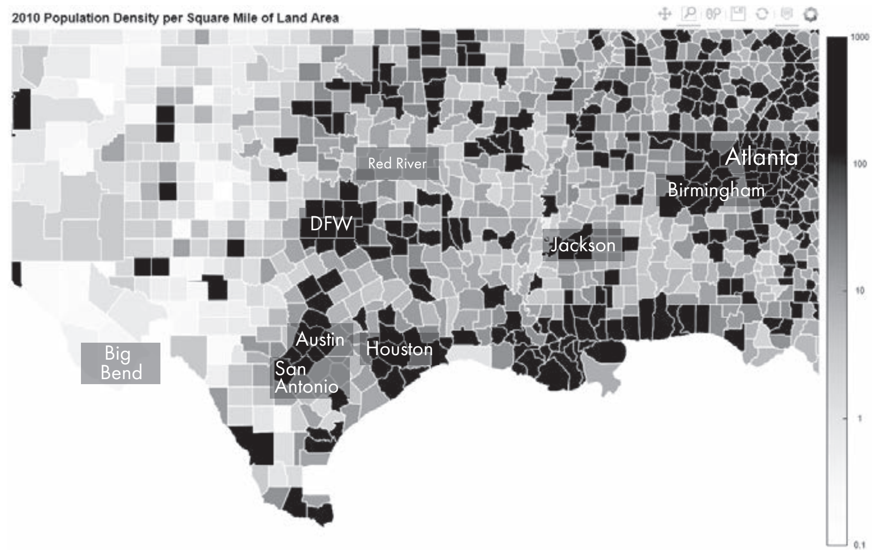


Рис. 11.12. Округа с населением более 65 человек на квадратную милю выделены черным

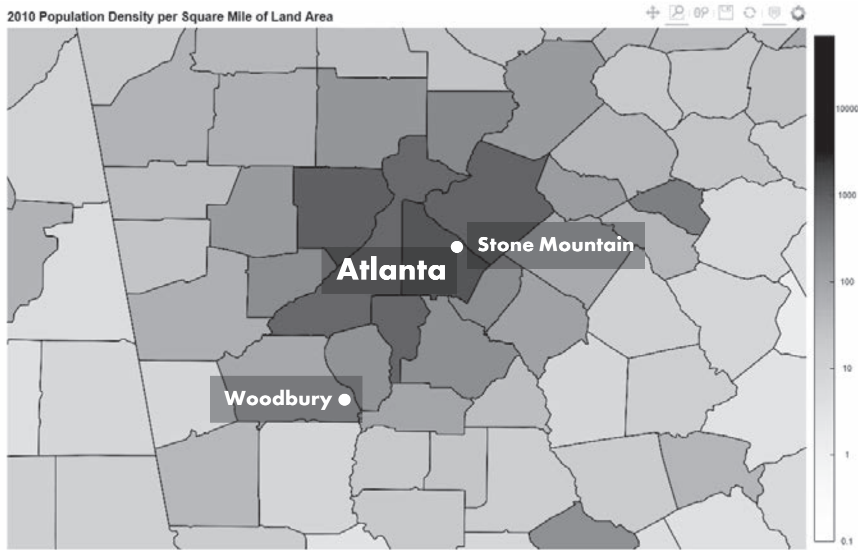


Рис. 11.13. Расположение Стоун-Маунтин и вымышленного города Вудбери

Стоун-Маунтин находится ближе 20 миль от Атланты в округе ДеКалб с населением 2586 человек на квадратную милю. Вудбери (в реальности город Сеноя) расположен всего в 35 милях от центра Атланты на границе округа Ковета с населением 289 человек на квадратную милю и округа Файетт с населением 549 человек на квадратную милю. Неудивительно, что у выживших было так много проблем. Если бы у них в группе был хотя бы один специалист по обработке данных...

Итоги

В этой главе вы научились работать с библиотекой анализа данных `pandas`, а также с модулями визуализации `bokeh` и `holoviews`. Вы осуществили первичную обработку данных, очистив их и связав информацию из разных источников.

Дополнительная литература

Статья «If the Zombie Apocalypse Happens, Scientists Say You Should Run for the Hills» («Ученые говорят, что если случится зомби-апокалипсис, то надо бежать в горы») (Business Insider, 2017) Кевина Лориа (Kevin Loria) описывает применение стандартных моделей заболеваемости к показателям увеличения числа зомби при зомби-апокалипсисе.

Статья «What to Consider When Creating Choropleth Maps» (Chartable, 2018) Лизы Шарлотты Рост (Lisa Charlotte Rost) содержит полезные рекомендации по составлению хороплетных карт. Найти ее можно на https://blog.datawrapper.de/choroplethmaps/.

Публикация Ларри Веру (Larry Weru) «Muddy America: Color Balancing the Election Map—Infographic» (STEM Lounge, 2019) демонстрирует способы увеличения количества полезных опций хороплетных карт на примере стандартной красно-синей карты, показывающей результаты выборов в США.

«Python Data Science Handbook: Essential Tools for Working with Data»[1] (O'Reilly Media, 2016), написанная Джеком Ван Дер Пласом (Jake VanderPlas), представляет подробный справочник важных инструментов Python для работы с данными, включая `pandas`.

«Beneath the Window: Early Ranch Life in the Big Bend Country» (Iron Mountain Press, 2003), написанная Патрицией Уилсон Клотьер (Patricia Wilson

[1] Вандер Плас Дж. «Python для сложных задач. Наука о данных и машинное обучение». СПб., издательство «Питер».

Clothier), — увлекательная история о жизни автора в начале XX века на обширном ранчо в округе Биг-Бенд штата Техас, еще до того, как эта территория стала Национальным парком. В книге автор размышляет на тему, как выжившие во время апокалипсиса могут обустроить жизнь в суровых природных условиях.

«Game Theory: Real Tips for SURVIVING a Zombie Apocalypse (7 Days to Die)» (The Game Theorists, 2016) — видеосюжет, посвященный лучшему в мире месту для проживания во время зомби-апокалипсиса. В отличие от «Ходячих мертвецов» сюжет предполагает, что вирус зомби может передаваться через москитов и клещей, на основе чего и происходит выбор места. Посмотреть сюжет можно онлайн.

Усложняем проект: отображение на карте изменения численности населения США

Правительство США ежегодно публикует данные переписи населения. На момент написания книги доступны промежуточные, не очень точные данные от 2019 года. Используйте один из их вариантов вместе с данными от 2010 года из проекта 15 для построения новой хороплетной карты, охватывающей изменение в населенности по округам в течение этого временного отрезка.

Подсказка: можно вычесть столбцы в датафрейме `pandas`, чтобы сгенерировать данные, отражающие разницу, как показано в примере ниже. Значения плотности населения на 2020 год представляют фиктивные данные.

```
>>> import pandas as pd
>>>
>>> # Генерируем набор данных о населении по округам:
>>> pop_2010 = {'county': ['Autauga', 'Baldwin', 'Barbour', 'Bibb'],
                'popl': [54571, 182265, 27457, 22915]}
>>> pop_2020 = {'county': ['Autauga', 'Baldwin', 'Barbour', 'Bibb'],
                'popl': [52910, 258321, 29073, 29881]}
>>>
>>> df_2010   =  pd.DataFrame(pop_2010)
>>> df_2020   =   pd.DataFrame(pop_2020)
>>> df_diff = df_2020.copy() # Копируем датафрейм 2020 года в новый датафрейм
>>> df_diff['diff'] = df_diff['popl'].sub(df_2010['popl']) # Вычитаем столбцы
                                                                popl
>>> print(df_diff.loc[:4, ['county', 'diff']])
    county   diff
0  Autauga  -1661
1  Baldwin  76056
2  Barbour   1616
3     Bibb   6966
```

12

Находимся ли мы в компьютерной симуляции?

В 2003 году философ Ник Бостром (Nick Bostrom) постулировал, что мы живем в компьютерной симуляции, созданной нашими высокоразвитыми предками, которые, вероятно, были сверхлюдьми. Сегодня многие ученые и думающие люди, включая Нила ДеГрассе Тайсона (Neil DeGrasse Tyson) и Илона Маска (Elon Musk), верят в то, что *гипотеза о симуляции* вполне может оказаться правдой. Она определенно объясняет, почему математики так элегантно описывают природу, почему наблюдатель оказывает влияние на наблюдаемые квантовые явления и почему мы одиноки во Вселенной.

Еще более странно, что *вы* можете оказаться единственным реальным объектом в этой симуляции. Возможно, вы — это «мозг в колбе», продуцирующий историческую симуляцию. Для повышения вычислительной эффективности эта симуляция может отображать только те вещи, с которыми вы в данный момент взаимодействуете. Когда вы входите внутрь и закрываете дверь, внешний мир может отключаться, подобно свету в холодильнике. Как можно это подтвердить или опровергнуть?

Ученые всерьез относятся к этой гипотезе: они ведут споры и публикуют работы о том, как можно провести тест для ее подтверждения. В этой главе мы попробуем ответить на данный вопрос с помощью подхода, предложенного физиками. Для этого мы создадим простой симулированный мир, после чего проанализируем его, чтобы найти доказательства, которые могли бы выдать симуляцию. При этом проект мы реализуем в обратном порядке, то есть сначала напишем код, а потом

продумаем стратегию решения задачи. Результат покажет, что даже простейшая модель может позволить глубоко заглянуть в природу нашего бытия.

Проект #16. Жизнь, Вселенная и пруд черепахи Йертл

Возможность симулировать реальность представляет собой не такую уж отдаленную мечту. Физики использовали мощнейшие в мире суперкомпьютеры для реализации такого замысла, смоделировав поведение субатомной частицы в масштабе нескольких фемтометров (10^{-15} м). Несмотря на то что эта симуляция — попытка представить всего лишь малую толику мироздания, ее невозможно отличить от реальности — как мы эту реальность понимаем.

Сразу хочу вас успокоить: супермощный компьютер или научная степень в физике для решения следующей задачи вам не потребуются. Все, что нужно, — это модуль `turtle`, программа рисования, созданная специально для детей. Мы уже использовали `turtle` для симуляции миссии «Аполлон-8» в главе 6. Здесь же с ее помощью попытаемся понять основные особенности компьютерных моделей. А затем разработаем стратегию, аналогичную той, которую физики планируют применить для проверки гипотезы симуляции.

ЗАДАЧА

Определить особенности компьютерной симуляции, которые могут быть обнаружены теми, кто в ней участвует.

Код симуляции пруда

Код pond_sim.py создает на основе `turtle` симуляцию пруда, в котором есть островок, дрейфующее бревно и плавающая туда-сюда черепаха по имени Йертл. Черепаха будет периодически плавать к бревну и обратно. Код можно скачать с сайта книги по адресу https://nostarch.com/real-world-python/.

Модуль `turtle` включен в Python, так что устанавливать его не придется. Более подробную информацию об этом модуле вы найдете в разделе «Использование модуля turtle» на с. 171.

Импортируем turtle, настраиваем экран и рисуем остров

Код листинга 12.1 импортирует `turtle`, настраивает объект `screen` на использование в качестве пруда и рисует островок, откуда Йертл будет обозревать свои владения.

Листинг 12.1. Импорт модуля turtle, отрисовка пруда и островка

pond_sim.py, часть 1

```
import turtle

pond = turtle.Screen()
pond.setup(600, 400)
pond.bgcolor('light blue')
pond.title("Yertle's Pond")

mud = turtle.Turtle('circle')
mud.shapesize(stretch_wid=5, stretch_len=5, outline=None)
mud.pencolor('tan')
mud.fillcolor('tan')
```

После импорта модуля `turtle` присваиваем объект `screen` переменной `pond`. С помощью метода `turtle setup()` настраиваем размер экрана в пикселях, после чего устанавливаем цвет фона `light blue`. Таблицы цветов `turtle` и их названия можно найти на разных сайтах, например здесь: https://trinket.io/docs/colors. Присваиваем экрану название, чтобы завершить создание пруда.

Далее создаем круглый островок, на котором будет загорать Йертл. С помощью класса `Turtle()` инстанцируем объект `turtle` под названием `mud`. Несмотря на то что в `turtle` есть метод для рисования кругов, проще передать конструктору аргумент `'circle'`, который создаст круг в `turtle`. Однако он слишком мал для островка, поэтому с помощью метода `shapesize()` мы его растягиваем. Завершаем создание острова, определяя его контуры и устанавливая цвета заполнения как `'tan'`.

Рисуем бревно, отверстия от сучка и черепаху Йертл

Листинг 12.2 рисует бревно с отверстием от сучка и черепаху Йертл, после чего перемещает Йертл так, чтобы та плыла от островка к бревну.

Листинг 12.2. Отрисовка бревна и черепахи с последующим ее перемещением

pond_sim.py, part 2

```
   SIDE = 80
   ANGLE = 90
   log = turtle.Turtle()
   log.hideturtle()
   log.pencolor('peru')
   log.fillcolor('peru')
   log.speed(0)
❶  log.penup()
   log.setpos(215, -30)
   log.lt(45)
   log.begin_fill()
```

```
❷ for _ in range(2):
        log.fd(SIDE)
        log.lt(ANGLE)
        log.fd(SIDE / 4)
        log.lt(ANGLE)
  log.end_fill()

  knot = turtle.Turtle()
  knot.hideturtle()
  knot.speed(0)
  knot.penup()
  knot.setpos(245, 5)
  knot.begin_fill()
  knot.circle(5)
  knot.end_fill()

  yertle = turtle.Turtle('turtle')
  yertle.color('green')
  yertle.speed(1) # Самая медленная
  yertle.fd(200)
  yertle.lt(180)
  yertle.fd(200)
❸ yertle.rt(176)
  yertle.fd(200)
```

Бревно мы изобразим как прямоугольник, поэтому начинаем с присваивания двух констант, `SIDE` и `ANGLE`. Первая задает длину бревна в пикселях, а вторая — угол в градусах, на который мы будем поворачивать черепаху-указатель в каждом углу прямоугольника.

По умолчанию все черепашки-указатели появляются в середине экрана в координатах (0, 0). Поскольку бревно мы помещаем в стороне, то после инстанцирования объекта `log` используем метод `hideturtle()`, чтобы сделать указатель невидимым. Таким образом, нам не придется наблюдать его передвижение по экрану в процессе рисования.

Делаем бревно коричневым, используя для цвета значение `peru`. Затем устанавливаем `speed` объекта на максимум (странно, но это `0`). Так мы ускорим его отрисовку. А для того чтобы не видеть путь, который он проделывает от центра экрана к своей точке, отключаем его отрисовку методом `penup()` ❶.

С помощью метода установки позиции `setpos()` помещаем бревно справа на экране. Затем поворачиваем объект на 45 градусов и вызываем метод `begin_fill()`.

Можно сократить код на несколько строк, нарисовав прямоугольник с помощью цикла `for` ❷. Мы сделаем два цикла, каждый рисует по две стороны фигуры. Устанавливаем для бревна ширину в 20 пикселей, поделив `SIDE` на `4`. После цикла вызываем `end_fill`, закрашивая его в коричневый цвет.

Сделаем изображение бревна более реалистичным, добавив отверстие от сучка, для этого используем указатель `knot`. Чтобы нарисовать это отверстие, вызываем метод `circle()` и передаем ему значение `5`, устанавливая радиус в 5 пикселей. Обратите внимание, что задавать цвет для отверстия не нужно, так как он по умолчанию будет черным.

Теперь нарисуем Йертл, владыку пруда. Йертл уже стара, поэтому установим скорость ее отрисовки на минимум, указав `1`. Далее зададим ее заплыв к бревну и обратно. У Йертл старческий маразм, и она забывает, что только что делала, поэтому обратно мы отправим ее спиной вперед, только теперь сделаем уклон, чтобы она смотрела не точно на восток ❸. Запускаем программу — у нас должен получиться результат, показанный на рис. 12.1.

Посмотрите на рисунок внимательно. Хотя эта симуляция и проста, она позволяет понять, пребываем ли мы, подобно черепахе Йертл, в компьютерной симуляции.

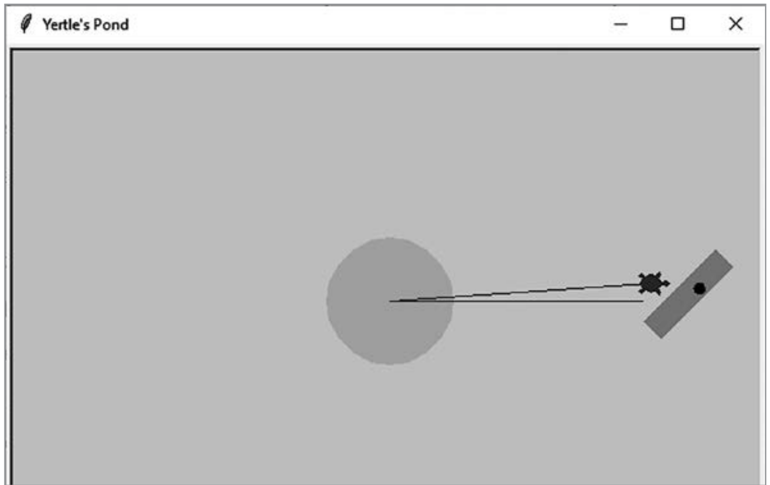


Рис. 12.1. Скриншот готовой симуляции

Следствия симуляции пруда

Из-за ограниченности вычислительных ресурсов всем компьютерным симуляциям требуется некий «каркас», на который они смогут «навесить» свою модель реальности. Независимо от того, назовем мы его сеткой, решеткой, матрицей или как-то иначе, он даст нам возможность распределять объекты в 2D- и 3D-пространстве и присваивать им свойства, например массу, температуру, цвет и пр.

В качестве системы координат, а также для сохранения свойств модуль `turtle` использует пиксели монитора. Их расположение определяет формы, например контуры бревна, а цвета пикселей позволяют отличить одну фигуру от другой.

Пиксели формируют *ортогональный* шаблон, подразумевающий, что строки и столбцы пикселей пересекаются под правильными углами. Несмотря на то что каждый пиксель имеет квадратную форму и очень мал, с помощью метода `turtle dot()` удается генерировать изображение, как в следующем сниппете:

```
>>> import turtle
>>>
>>> t = turtle.Turtle()
>>> t.hideturtle()
>>> t.penup()
>>>
>>> def dotfunc(x, y):
        t.setpos(x, y)
        for _ in range(10):
                t.dot()
                t.fd(10)

>>> for i in range(0, 100, 10):
        dotfunc(0, -i)
```

В результате мы получим шаблон, как на рис. 12.2.

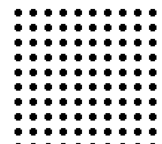


Рис. 12.2. Ортогональная сетка из черных точек, представляющих центры квадратных пикселей

В мире `turtle` пиксели, как и настоящие атомы, неделимы. Линия не может быть короче одного пикселя. Движение реализуется только целым числом пикселей (хотя значение с плавающей точкой ошибку не вызовет). Минимальный объект тоже представлен одним пикселем.

Следствие из этого таково: сетка в симуляции определяет наименьший признак, который мы можем наблюдать. Поскольку человечеству уже известны невероятно крохотные субатомные частицы, то сетка, если мы находимся в симуляции, должна быть чрезвычайно мелкой. Это и заставляет многих ученых сомневаться в действительности такой гипотезы, так как для реализации подобной симуляции потребовался бы чудовищный объем компьютерной памяти. И все же, кто знает, какими возможностями обладали наши далекие предки или инопланетяне?

Помимо ограничения размера объектов, сетка в симуляции может определять предпочтительную ориентацию, или *анизотропию*, материи космоса. Анизотропия характеризует различие свойств среды в зависимости от направления. Например, дерево проще расколоть вдоль волокон, чем поперек. Если внимательно посмотреть на траектории движения Йертл в симуляции (рис. 12.3), то мы увидим признаки анизотропии. Верхняя, слегка наклонная траектория неровная, а нижняя — с запада на восток — идеально прямая.

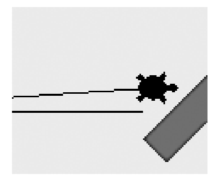


Рис. 12.3. Траектории движения черепахи

Изображение неортогональной линии на ортогональной сетке выглядит некрасиво. Но здесь дело не только в эстетике. Перемещение вдоль x или y требует прибавления или вычитания только целых значений (рис. 12.4, слева). Перемещение под углом требует тригонометрического вычисления частичного смещения в направлениях x и y (рис. 12.4, справа).

Для компьютера математические вычисления равносильны работе, то есть можно заключить, что движение под углом требует больше энергии. Замерив время вычислений для двух траекторий на рис. 12.4, мы можем получить разницу в затраченной энергии.

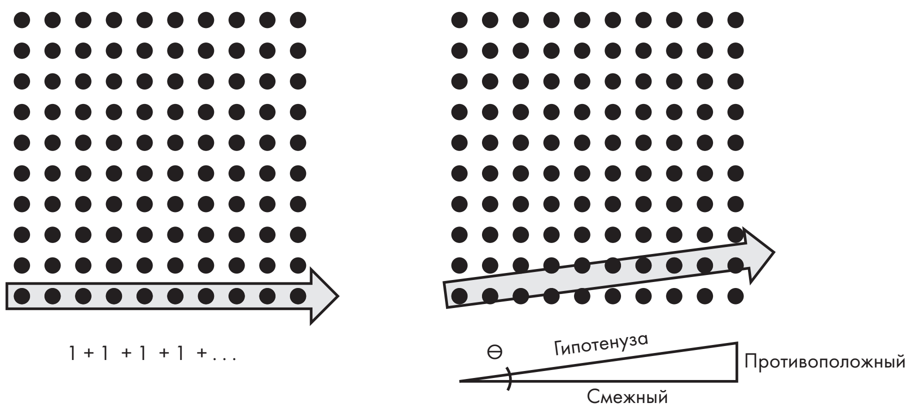


Рис. 12.4. Перемещение вдоль строк или столбцов (*слева*) требует более простых арифметических вычислений, чем перемещение с пересечением строки или столбца (*справа*)

Измерение затрат на пересечение строк или столбцов сетки

Чтобы вычислить разницу во времени при рисовании линии вдоль строки или столбца и линии, пересекающей их, необходимо нарисовать две линии одинаковой длины. Но при этом нужно помнить, что `turtle` работает только с целыми

числами. Придется найти угол, для которого все стороны треугольника — катеты и гипотенуза на рис. 12.4 — будут представлены целыми числами. Так мы будем знать, что наклонная линия имеет ту же длину, что и прямая.

Для нахождения этих углов можно использовать *пифагорову тройку*, набор положительных целых чисел a, b и c, которые удовлетворяют квадратному уравнению $a^2 + b^2 = c^2$. Известна тройка 3-4-5, но нам может потребоваться более длинная линия, чтобы время, затраченное на рисование линии, не оказалось меньше точности измерения часов компьютера. Удобно, что другие тройки значений можно также найти онлайн. Неплохим вариантом будет, например, сочетание 62-960-962, так как линия получится достаточно длинной и при этом поместится на экране `turtle`.

Код для сравнения линий

Для рисования диагональной и прямой линий мы воспользуемся `turtle` (листинг 12.3). Первую линию мы рисуем параллельно оси x (то есть запад-восток), а вторую под небольшим углом к ней. Верный угол можно вычислить с помощью тригонометрии; в данном случае это 3.695220532 градуса. Код многократно повторяет цикл `for` и записывает время, затраченное на каждую линию, используя модуль `time`. В итоге сравнивается среднее время для обеих линий.

Средние мы берем, потому что CPU компьютера непрерывно обрабатывает множество процессов. Операционная система планирует эти процессы внутренне, выполняя один и откладывая другой, пока ресурс ввода/вывода не станет доступен. Следовательно, сложно зарегистрировать *абсолютное* время выполнения заданной функции. Вычисление среднего значения времени для множества выполнений учитывает это обстоятельство.

Код, line_compare.py, можно скачать с сайта книги.

Листинг 12.3. Отрисовка прямой и наклонной линий и регистрация затраченного на каждую линию времени

line_compare.py

```
from time import perf_counter
import statistics
import turtle

turtle.setup(1200, 600)
screen = turtle.Screen()

ANGLES = (0, 3.695220532) # В градусах
NUM_RUNS = 20
SPEED = 0
for angle in ANGLES:
  ❶ times = []
```

```
        for _ in range(NUM_RUNS):
            line = turtle.Turtle()
            line.speed(SPEED)
            line.hideturtle()
            line.penup()
            line.lt(angle)
            line.setpos(-470, 0)
            line.pendown()
            line.showturtle()
          ❷ start_time = perf_counter()
            line.fd(962)
            end_time = perf_counter()
            times.append(end_time - start_time)

    line_ave = statistics.mean(times)
    print("Angle {} degrees: average time for {} runs at speed {} = {:.5f}"
        .format(angle, NUM_RUNS, SPEED, line_ave))
```

Начинаем с импорта *счетчика производительности* `perf_counter` из модуля `time`. Эта функция возвращает время в секундах в виде значения с плавающей точкой. При этом ответ она дает более точный, чем `time.clock()`, которую заменила, начиная с Python 3.8.

Далее импортируем модуль `statistics`, с помощью которого вычислим среднее значение по множеству запусков симуляции. После импортируем `turtle` и настраиваем `turtle screen`. Вы можете подстроить экран под свой монитор, но помните, что у вас должна быть возможность видеть линию длиной 962 пикселя.

Теперь присваиваем значения ключей для симуляции. Помещаем углы для прямой и наклонной линий в кортеж `ANGLES`, после чего присваиваем переменную для хранения числа циклов `for` и скорости отрисовки линии.

Начинаем перебор углов в кортеже `ANGLES`. Создаем пустой список для хранения измеренных значений времени ❶, после чего настраиваем объект `turtle`, как делали это ранее. Поворачиваем объект `turtle` влево на значение `angle` и с помощью `setpos()` перемещаем его к левому краю экрана.

Перемещаем указатель-черепаху вперед на 962 пикселя, вписывая команду между вызовами `perf_counter()`, чтобы замерить время этого перемещения ❷. Вычитаем итоговое время из начального и заносим результат в список `times`.

Завершаем процесс, используя функцию `statistics.mean()` для вычисления среднего времени отрисовки каждой линии. Выводим результат с точностью до пяти десятичных знаков. После выполнения программы результат должен выглядеть, как на рис. 12.5.

Так как мы использовали пифагорову тройку, наклонная линия действительно завершается в пикселе, а не просто привязывается к ближайшему из них. Это

гарантирует одинаковую длину обеих линий и точность сравнения времени их отрисовки.

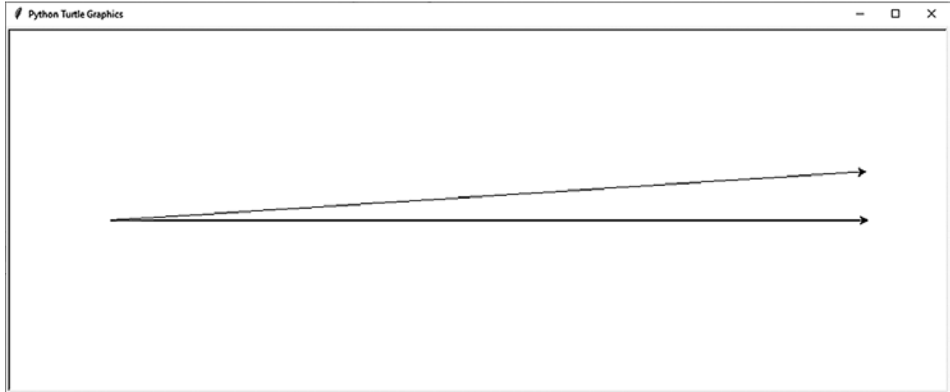


Рис. 12.5. Результирующее изображение на экране turtle для line_compare.py

Результаты

Если вы нарисуете каждую линию 500 раз и сравните результаты, то увидите, что на наклонную линию тратится примерно в 2.4 раза больше времени.

```
Angle 0 degrees: average time for 500 runs at speed 0 = 0.06492
Angle 3.695220532 degrees: average time for 500 runs at speed 0 = 0.15691
```

У вас значения времени наверняка будут немного отличаться, поскольку на скорость выполнения влияют и другие программы, параллельно выполняемые на вашем компьютере. Как я уже отмечал, всеми этими процессами руководит планировщик задач CPU, обеспечивая быстродействие системы, а также эффективность и равномерность распределения ее ресурсов.

Если повторить эксперимент 1000 раз, то результаты должны оказаться такими же. (Если решите это проделать, то не забудьте прежде запастись чашечкой кофе и вкусностями.) Наклонная линия требует примерно в 2.7 раза больше времени.

```
Angle 0 degrees: average time for 1000 runs at speed 0 = 0.10911
Angle 3.695220532 degrees: average time for 1000 runs at speed 0 = 0.29681
```

Здесь мы выполняли короткую функцию с высокой скоростью отрисовки. Если вас беспокоит тот факт, что `turtle` с целью повышения скорости выполняет оптимизацию, пренебрегая точностью, то можете уменьшить значение скорости и выполнить программу еще раз. При установке значения `speed = 6` на рисование

наклонной линии потребуется примерно в 2.6 раза больше времени, что очень близко к результату при выполнении на максимальной скорости.

```
Angle 0 degrees: average time for 500 runs at speed 6 = 1.12522
Angle 3.695220532 degrees: average time for 500 runs at speed 6 = 2.90180
```

Очевидно, что перемещение с пересечением пиксельной сетки требует больше работы, чем движение вдоль нее.

Стратегия

Задача этого проекта — определить, есть ли у симулированных персонажей, быть может, и у нас с вами, возможность найти свидетельства своего существования в симуляции. Во-первых, если мы живем в симуляции, то ее сетка невероятно мала, так как мы можем наблюдать даже субатомные частицы. Во-вторых, если эти мелкие частицы пересекают сетку под углом, то мы должны ожидать возникновения вычислительного сопротивления, которое будет проявляться как что-то измеримое — как потеря энергии, рассеяние частиц, снижение скорости или другие аналогичные эффекты.

В 2012 году физик Сайлас Р. Бин (Silas R. Beane) из Университета Бонна совместно с Зохре Давуди (Zohreh Davoudi) и Мартином Дж. Севиджем (Martin J. Savage) из Университета Вашингтона опубликовали работу, в которой доказывали именно эту идею. По мнению авторов, если законы физики, которые являются непрерывными, наложить на дискретную решетку, то ее шаг может вызвать ограниченность физических процессов.

Они предложили изучить этот вопрос с помощью наблюдения *космических лучей сверхвысокой энергии* (ultra-high energy cosmic rays, UHECR). UHECR — это самые быстрые частицы во Вселенной, которые по мере роста своей энергии начинают испытывать влияние все более мелких характеристик пространства. Однако для этих частиц определена максимально достижимая энергетическая граница. Она известна как предел Грайзена — Зацепина — Кузьмина (предел ГЗК) и подтверждена экспериментально в 2007 году. Эта граница согласуется с тем, какой вид ограничения может вызывать сетка симуляции. Подобное условие также должно способствовать перемещению UHECR вдоль осей сетки и рассеянию частиц, пытающихся двигаться поперек.

Неудивительно, что для реализации такого опыта существует множество преград. Подобные частицы редки, а аномальное поведение может не быть столь очевидным. Если шаг сетки окажется намного меньше 10^{-12} фемтометров, то отклонений мы, возможно, и не заметим. К тому же, сетки может вообще не существовать, по крайней мере в привычном для нас понимании, так как использованная для ее реализации технология наверняка намного превосходит

наш уровень знаний. И, как заметил в 2019 году философ Престон Грин (Preston Greene), для реализации подобного проекта может существовать моральное препятствие. Имеется в виду, если мы живем в компьютерной симуляции, то обнаружение нами сего факта может привести к ее завершению.

Итоги

Написать код для симуляции мира черепахи Йертл было просто. Но программы в большей степени используются для решения задач, и даже небольшой объем проделанной нами работы дает необходимый опыт. Нет, мы не совершили прорыв в области изучения космических лучей, но начали разговор в правильном русле. Сама идея о том, что компьютерной симуляции требуется сетка, которая могла бы отражать наблюдаемые сигнатуры Вселенной, далеко не мелочь.

В книге «Гарри Поттер и Дары Смерти» Гарри спрашивает волшебника Дамблдора: «Ответьте мне на один последний вопрос. Все это реально? Или просто происходит у меня в голове?» На что тот отвечает: «Конечно же, это происходит в твоей голове, Гарри, но почему сей факт должен означать, что это нереально?»

Даже если наш мир и не находится на «фундаментальном уровне реальности», как заявляет Ник Бостром, то это никак не мешает нам наслаждаться решением подобных задач. Декарт, будь он жив сегодня, мог бы выразиться так: «Я пишу код, следовательно, я существую». Вперед!

Дополнительная литература

В статье «Are We Living in a Simulated Universe? Here's What Scientists Say» (NBC News, 2019) автор Дэн Фальк (Dan Falk) дает обзор гипотез симулированной реальности.

«Neil deGrasse Tyson Says 'It's Very Likely' the Universe Is a Simulation» (ExtremeTech, 2016) Грэхема Темплтона (Graham Templeton) содержит видео одной из ежегодных дискуссий памяти Айзека Азимова, которую проводит астрофизик Нил деГрассе Тайсон (Neil deGrasse Tyson), — ученые из разных областей дискутируют на тему, живем ли мы в симуляции.

«Are We Living in a Computer Simulation? Let's Not Find Out» (New York Times, 2019) — статья Престона Грина (Preston Greene), представляющая философские доводы против изучения гипотезы симуляции.

«We Are Not Living in a Simulation. Probably» (Fast Company, 2018) Гленна Макдональда (Glenn McDonald) посвящена обсуждению того, что Вселенная слишком велика и детализирована для компьютерной симуляции.

Дополнение

В жизни никогда не хватает времени для всего, что мы хотим сделать, а в случае написания книги это актуально вдвойне. Приведенные далее усложненные проекты можно рассматривать как призраки еще не написанных глав. Для их завершения не хватило времени (а в некоторых случаях их не удалось даже начать), но для вас это может и не стать проблемой. Как обычно, решений для более сложных задач я не предлагаю — не факт, что они вам вообще понадобятся.

Это реальный мир, мой дорогой друг, и ты к нему готов.

Усложняем проект: поиск безопасного места в космосе

Всемирно известный роман «Мир-Кольцо» 1970 года познакомил читателей с кукловодами Пирсона, разумными инопланетными травоядными существами. Будучи стадными животными, кукловоды чрезвычайно трусливы и осторожны. Узнав о том, что ядро Млечного Пути взорвалось и радиация убьет их через 20 000 лет, они решили покинуть галактику.

В этом проекте вы станете членом дипломатической команды посла кукловодов в 29 веке. Ваша задача — выбрать один из сопредельных Соединенных штатов, который кукловоды сочтут достаточно безопасным для своего посольства. Вам потребуется оценить каждый штат на предмет наличия природных угроз, таких как землетрясения, торнадо, вулканы и ураганы, после чего представить послу обобщенную карту. Не беспокойтесь, что используемые вами данные будут устаревшими на несколько сотен лет. Просто представьте, что эти данные актуальны в 2850 году н. э.

Данные по землетрясениям можно найти на https://earthquake.usgs.gov/earthquakes/feed/v1.0/csv.php/. Обозначьте точками эпицентры толчков с магнитудой 6.0 балла и выше.

Данные о торнадо можете указать в виде среднего количества в год для каждого штата (информация — на https://www.ncei.noaa.gov/access/monitoring/monthly-report/tornadoes/). Используйте хороплетную карту так же, как в главе 11.

Список опасных вулканов находится в таблице 2 обновлений от 2018 года, предоставленных Геологической службой США (https://pubs.usgs.gov/sir/2018/5140/sir20185140.pdf). Нанесите их на карту в виде точек, но присвойте им цвет или форму, отличающие их от данных по землетрясениям. При этом не обращайте внимания на выбросы пепла из Йеллоустоуна. Предположите, что эксперты, занимающиеся наблюдением за этим супервулканом, могут спрогнозировать извержение с достаточным запасом времени, чтобы посол успел покинуть планету.

Для получения данных о траекториях движения ураганов обратитесь к сайту Национального управления океанических и атмосферных исследований (https://coast.noaa.gov/digitalcoast/data/), где отыщите «Historical Hurricane Tracks». Скачайте и нанесите на карту штормовые участки категории 4 и выше.

Попытайтесь мыслить как кукловод и выберите на основе полученной карты подходящий для их посольства штат. Возможно, придется также проигнорировать торнадо. Америка — это очень опасное место!

Усложняем проект: а вот и Солнце

В 2018 году 13-летняя Джорджия Хатчинсон из Вудсайда, штат Калифорния, выиграла $25 000 в международном конкурсе Broadcom Masters, который проводился среди учащихся средних школ в рамках программы STEM (science, technology, engineering, mathematics — наука, технология, инженерия и математика). Ее работа «Designing a Data-Driven Dual-Axis Solar Tracker» (Проектирование двухосевого солнечного трекера, управляемого данными) удешевит солнечные панели и сделает их более доступными, исключив необходимость использования дорогостоящих световых сенсоров.

В основе работы такого солнечного трекера лежит идея о том, что нам уже известно расположение Солнца в любой момент времени относительно любой точки поверхности Земли. На основе открытых данных Национального управления океанических и атмосферных явлений трекер непрерывно определяет положение Солнца, а солнечные панели наклоняются оптимальным образом для максимальной эффективности сбора его энергии.

Напишите программу Python, вычисляющую положение Солнца на основе выбранной вами локации. Для начала ознакомьтесь со статьей «Position of the Sun» в Википедии на https://en.wikipedia.org/wiki/Position_of_the_Sun.

Усложняем проект: взгляд глазами собаки

Используйте свои знания в области компьютерного зрения, чтобы написать программу Python, получающую изображение и имитирующую то, как его должна видеть собака. Для начала ознакомьтесь с информацией на страницах https://www.akc.org/expert-advice/health/are-dogs-color-blind/ и https://dog-vision.andraspeter.com/.

Усложняем проект: кастомизированный поиск слов

Представьте, что ваша бабушка увлекается головоломками «Найди слово». Используйте Python, чтобы к ее дню рождения спроектировать и вывести

специальные варианты этих головоломок, где будут фигурировать имена членов семьи, персонажи старых ТВ-шоу вроде «Мэтлока» и «Коломбо» или названия знакомых ей медицинских препаратов. Пусть слова выводятся по горизонтали, вертикали и диагонали.

Усложняем проект: оптимизация праздничного показа слайдов

Ваша супруга, брат, отец, лучший друг или просто знакомый устраивает праздничный ужин, и вам поручено организовать показ слайдов. У вас есть куча фотографий в облаке, на многих из них присутствует виновник торжества, но имена файлов содержат лишь дату и время, когда была сделана фотография, не поясняя ее содержимое. Похоже, вам придется провести всю субботу за их сортировкой.

Но не стоит отчаиваться, ведь вы научились работать с распознаванием лиц. Все, что вам нужно, — это найти несколько обучающих изображений и написать небольшой фрагмент кода.

Сначала выберите из своей цифровой коллекции фото виновника торжества. Затем напишите на Python программу, которая будет выполнять поиск по каталогам, находить фотографии с этим человеком и копировать их в отдельную папку для просмотра. В процессе обучения не забудьте включить фото в фас и в профиль, а также добавить обученный для профилей каскад Хаара.

Усложняем проект: что за сложную паутину мы плетем

Используйте Python и модуль `turtle` для симуляции плетущего паутину паука. Вспомогательную информацию по ее построению можете найти на странице https://www.brisbaneinsects.com/brisbane_weavers/index.htm.

Усложняем проект: идем вещать с горы

«Какая ближайшая гора к Хьюстону, штат Техас?» На этот, казалось бы, простой вопрос не столь просто ответить. Во-первых, нужно взять в расчет горы в Мексике, а также в США. Во-вторых, единого утвержденного понятия горы не существует.

Чтобы эту задачу несколько упростить, используйте одно из определений *горной местности*, установленное Программой ООН по защите окружающей среды. Найдите участки с высотой не менее 2500 м (8200 футов), которые вы будете считать горами. Вычислите расстояние до них от центра Хьюстона, чтобы определить ближайшую.

Решения для практических проектов

Это приложение содержит решения для практических проектов каждой главы. Вы можете скачать их с сайта книги https://nostarch.com/real-world-python/.

Глава 2. Определение авторства с помощью стилометрии

Охота на собаку Баскервилей с помощью распределения

practice_hound_dispersion.py

```
"""Создаем график распределения с помощью NLP (nltk)."""
import nltk
import file_loader

corpus = file_loader.text_to_string('hound.txt')
tokens = nltk.word_tokenize(corpus)
tokens = nltk.Text(tokens) # Обертка NLTK для автоматического анализа текста.
dispersion = tokens.dispersion_plot(['Holmes',
                                     'Watson',
                                     'Mortimer',
                                     'Henry',
                                     'Barrymore',
                                     'Stapleton',
                                     'Selden',
                                     'hound'])
```

Тепловая карта пунктуации

practice_heatmap_semicolon.py

```
"""Создаем тепловую карту пунктуации."""
import math
from string import punctuation
import nltk
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import seaborn as sns

# Устанавливаем seaborn через pip install seaborn.

PUNCT_SET = set(punctuation)

def main():
    # Загружаем файлы текстов в словарь по автору.
    strings_by_author = dict()
    strings_by_author['doyle'] = text_to_string('hound.txt')
    strings_by_author['wells'] = text_to_string('war.txt')
    strings_by_author['unknown'] = text_to_string('lost.txt')

# Токенизируем строки текста, сохраняя только знаки препинания.
punct_by_author = make_punct_dict(strings_by_author)

# Преобразуем знаки препинания в численные значения и отображаем тепловые
  карты.
for author in punct_by_author:
    heat = convert_punct_to_number(punct_by_author, author)
    arr = np.array((heat[:6561])) # Обрезаем до наибольшего размера
                                    для квадратной матрицы
    arr_reshaped = arr.reshape(int(math.sqrt(len(arr))),
                               int(math.sqrt(len(arr))))
    fig, ax = plt.subplots(figsize=(7, 7))
    sns.heatmap(arr_reshaped,
                cmap=ListedColormap(['blue', 'yellow']),
                square=True,
                ax=ax)
    ax.set_title('Heatmap Semicolons {}'.format(author))
plt.show()

def text_to_string(filename):
    """Считываем файл текста и возвращаем строку."""
    with open(filename) as infile:
        return infile.read()

def make_punct_dict(strings_by_author):
    """Возвращаем словарь токенизированных знаков препинания по корпусу
    и автору."""
```

```
    punct_by_author = dict()
    for author in strings_by_author:
        tokens = nltk.word_tokenize(strings_by_author[author])
        punct_by_author[author] = ([token for token in tokens
                                    if token in PUNCT_SET])
        print("Number punctuation marks in {} = {}"
              .format(author, len(punct_by_author[author])))
    return punct_by_author

def convert_punct_to_number(punct_by_author, author):
    """Возвращаем список знаков препинания, преобразованных в численные
    значения."""
    heat_vals = []
    for char in punct_by_author[author]:
        if char == ';':
            value = 1
        else:
            value = 2
        heat_vals.append(value)
    return heat_vals

if __name__ == '__main__':
    main()
```

Глава 4. Отправка суперсекретных сообщений с помощью книжного шифра

Составление графика символов

practice_barchart.py

```
"""Рисуем столбчатую диаграмму символов из файла текста."""
import sys
import os
import operator
from collections import Counter
import matplotlib.pyplot as plt

def load_file(infile):
    """Считываем и возвращаем файл текста в виде строки символов в нижнем
    регистре."""
    with open(infile) as f:
        text = f.read().lower()
    return text

def main():
    infile = 'lost.txt'
    if not os.path.exists(infile):
        print("File {} not found. Terminating.".format(infile),
              file=sys.stderr)
        sys.exit(1)
```

```
    text = load_file(infile)

    # Создаем диаграмму символов в тексте и их частотности.
      char_freq = Counter(text)
    char_freq = Counter(text)
    char_freq_sorted = sorted(char_freq.items(),
                              key=operator.itemgetter(1), reverse=True)
    x, y = zip(*char_freq_sorted) # * для разделения ('распаковки')
                                    итераторов.
    fig, ax = plt.subplots()
    ax.bar(x, y)
    fig.show()

if __name__ == '__main__':
    main()
```

Отправка секретов шифром времен Второй мировой войны

practice_WWII_words.py

```
"""Книжный шифр, использующий роман "Затерянный мир".
Для слов, отсутствующих в книге, пишется их первая буква.
Отмечайте 'режим первой буквы', заключая слова между чередующимися
'a a' и 'the the'.

Реализация: Эрик Т. Мортенсон
"""
import sys
import os
import random
import string
from collections import defaultdict, Counter

def main():
    message = input("Enter plaintext or ciphertext: ")
    process = input("Enter 'encrypt' or 'decrypt': ")
    shift = int(input("Shift value (1-365) = "))
    infile = input("Enter filename with extension: ")

    if not os.path.exists(infile):
        print("File {} not found. Terminating.".format(infile),
              file=sys.stderr)
        sys.exit(1)
    word_list = load_file(infile)
    word_dict = make_dict(word_list, shift)
    letter_dict = make_letter_dict(word_list)

    if process == 'encrypt':
        ciphertext = encrypt(message, word_dict, letter_dict)
        count = Counter(ciphertext)
        encryptedWordList = []
        for number in ciphertext:
            encryptedWordList.append(word_list[number - shift])
```

```
        print("\nencrypted word list = \n {} \n"
            .format(' '.join(encryptedWordList)))
        print("encrypted ciphertext = \n {}\n".format(ciphertext))

        # Проверяем шифрование, расшифровывая криптограмму.
        print("decrypted plaintext = ")
        singleFirstCheck = False
        for cnt, i in enumerate(ciphertext):
            if word_list[ciphertext[cnt]-shift] == 'a' and \
               word_list[ciphertext[cnt+1]-shift] == 'a':
                continue
            if word_list[ciphertext[cnt]-shift] == 'a' and \
               word_list[ciphertext[cnt-1]-shift] == 'a':
                singleFirstCheck = True
                continue
            if singleFirstCheck == True and cnt<len(ciphertext)-1 and \
               word_list[ciphertext[cnt]-shift] == 'the' and \
                            word_list[ciphertext[cnt+1]-shift] == 'the':
                continue
            if singleFirstCheck == True and \
               word_list[ciphertext[cnt]-shift] == 'the' and \
                            word_list[ciphertext[cnt-1]-shift] == 'the':
                singleFirstCheck = False
                print(' ', end='', flush=True)
                continue
            if singleFirstCheck == True:
               print(word_list[i - shift][0], end = '', flush=True)
            if singleFirstCheck == False:
               print(word_list[i - shift], end=' ', flush=True)

    elif process == 'decrypt':
        plaintext = decrypt(message, word_list, shift)
        print("\ndecrypted plaintext = \n {}".format(plaintext))

def load_file(infile):
    """Считываем и возвращаем текстовый файл в виде списка слов в нижнем
    регистре."""
    with open(infile, encoding='utf-8') as file:
        words = [word.lower() for line in file for word in line.split()]
        words_no_punct = ["".join(char for char in word if char not in \
                            string.punctuation) for word in words]
    return words_no_punct

def make_dict(word_list, shift):
    """Возвращаем словарь символов в качестве ключей и смещенные индексы
    в качестве значений."""
    word_dict = defaultdict(list)
    for index, word in enumerate(word_list):
        word_dict[word].append(index + shift)
    return word_dict
```

```
def make_letter_dict(word_list):
    firstLetterDict = defaultdict(list)
    for word in word_list:
        if len(word) > 0:
            if word[0].isalpha():
                firstLetterDict[word[0]].append(word)
    return firstLetterDict

def encrypt(message, word_dict, letter_dict):
    """Возвращаем список индексов, представляющих символы в сообщении."""
    encrypted = []
    # Удаляем пунктуацию из слов сообщения.
    messageWords = message.lower().split()
    messageWordsNoPunct = ["".join(char for char in word if char not in \
                          string.punctuation) for word in messageWords]
    for word in messageWordsNoPunct:
        if len(word_dict[word]) > 1:
            index = random.choice(word_dict[word])
        elif len(word_dict[word]) == 1: # Random.choice дает сбой, если есть
                                          всего 1 выбор.
            index = word_dict[word][0]
        elif len(word_dict[word]) == 0: # Слова нет в word_dict.
            encrypted.append(random.choice(word_dict['a']))
            encrypted.append(random.choice(word_dict['a']))

            for letter in word:
                if letter not in letter_dict.keys():
                    print('\nLetter {} not in letter-to-word dictionary.'
                          .format(letter), file=sys.stderr)
                    continue
                if len(letter_dict[letter])>1:
                    newWord =random.choice(letter_dict[letter])
                else:
                    newWord = letter_dict[letter][0]
                if len(word_dict[newWord])>1:
                    index = random.choice(word_dict[newWord])
                else:
                    index = word_dict[newWord][0]
            encrypted.append(index)

            encrypted.append(random.choice(word_dict['the']))
            encrypted.append(random.choice(word_dict['the']))
            continue
        encrypted.append(index)
    return encrypted

def decrypt(message, word_list, shift):
    """Расшифровываем строку криптограммы и возвращаем строку слов в виде
    открытого текста.

    Здесь отображается, как выглядит открытый текст до извлечения
    первых букв.
    """
    plaintextList = []
```

```
    indexes = [s.replace(',', '').replace('[', '').replace(']', '')
               for s in message.split()]
    for count, i in enumerate(indexes):
        plaintextList.append(word_list[int(i) - shift])
    return ' '.join(plaintextList)

def check_for_fail(ciphertext):
    """Возвращаем True, если криптограмма содержит повторы ключей."""
    check = [k for k, v in Counter(ciphertext).items() if v > 1]
    if len(check) > 0:
        print(check)
        return True

if __name__ == '__main__':
    main()
```

Глава 5. Поиск Плутона

Представление орбитальной траектории

practice_orbital_path.py

```
import os
from pathlib import Path
import cv2 as cv

PAD = 5 # Игнорировать пиксели на таком расстоянии от края изображения.

def find_transient(image, diff_image, pad):
    """Получает изображение, изображение отличий и значение отступа
    в пикселях, на что возвращает булево значение и расположение maxVal
    на изображении отличий, исключая отступ по краям.
    Обводит на изображении кругом maxVal."""
    transient = False
    height, width = diff_image.shape
    cv.rectangle(image, (PAD, PAD), (width - PAD, height - PAD), 255, 1)
    minVal, maxVal, minLoc, maxLoc = cv.minMaxLoc(diff_image)
    if pad < maxLoc[0] < width - pad and pad < maxLoc[1] < height - pad:
        cv.circle(image, maxLoc, 10, 255, 0)
        transient = True
    return transient, maxLoc

def main():
    night1_files = sorted(os.listdir('night_1_registered_transients'))
    night2_files = sorted(os.listdir('night_2'))
    path1 = Path.cwd() / 'night_1_registered_transients'
    path2 = Path.cwd() / 'night_2'
    path3 = Path.cwd() / 'night_1_2_transients'

    # Все изображения должны быть сделаны с похожей экспозицией и иметь
      одинаковый размер.
    for i, _ in enumerate(night1_files[:-1]): # Убрать негативное изображение
        img1 = cv.imread(str(path1 / night1_files[i]), cv.IMREAD_GRAYSCALE)
        img2 = cv.imread(str(path2 / night2_files[i]), cv.IMREAD_GRAYSCALE)
```

```
        # Получаем абсолютную разницу между изображениями.
        diff_imgs1_2 = cv.absdiff(img1, img2)
        cv.imshow('Difference', diff_imgs1_2)
        cv.waitKey(2000)

        # Копируем изображение отличий, после чего находим и обводим самый
яркий пиксель.
        temp = diff_imgs1_2.copy()
        transient1, transient_loc1 = find_transient(img1, temp, PAD)

        # Рисуем черный круг на временном изображении, чтобы скрыть самое
яркое пятно.
        cv.circle(temp, transient_loc1, 10, 0, -1)

        # Получаем положение самого яркого пикселя и обводим его на входном
изображении.
        transient2, transient_loc2 = find_transient(img1, temp, PAD)

        if transient1 or transient2:
            print('\nTRANSIENT DETECTED between {} and {}\n'
                .format(night1_files[i], night2_files[i]))
            font = cv.FONT_HERSHEY_COMPLEX_SMALL
            cv.putText(img1, night1_files[i], (10, 25),
                       font, 1, (255, 255, 255), 1, cv.LINE_AA)
            cv.putText(img1, night2_files[i], (10, 55),
                       font, 1, (255, 255, 255), 1, cv.LINE_AA)
            if transient1 and transient2:
                cv.line(img1, transient_loc1, transient_loc2, (255, 255, 255),
                        1, lineType=cv.LINE_AA)

            blended = cv.addWeighted(img1, 1, diff_imgs1_2, 1, 0)
            cv.imshow('Исследовано', blended)
            cv.waitKey(2500) # Удерживает окно открытым в течение 2.5 секунды.

            out_filename = '{}_DECTECTED.png'.format(night1_files[i][:-4])
            cv.imwrite(str(path3 / out_filename), blended) # Будет
                                                             перезаписано!

        else:
            print('\nNo transient detected between {} and {}\n'
                  .format(night1_files[i], night2_files[i]))

if __name__ == '__main__':
    main()
```

В чем разница?

В этом практическом проекте используются две программы, Practice_montage_aligner.py и practice_montage_difference_finder.py. Выполнять их нужно в предложенном порядке.

practice_montage_aligner.py

practice_montage_aligner.py

```
import numpy as np
import cv2 as cv

MIN_NUM_KEYPOINT_MATCHES = 150

img1 = cv.imread('montage_left.JPG', cv.IMREAD_COLOR) # queryImage
img2 = cv.imread('montage_right.JPG', cv.IMREAD_COLOR) # trainImage
img1 = cv.cvtColor(img1, cv.COLOR_BGR2GRAY) # Преобразуем в полутона.
img2 = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)

orb = cv.ORB_create(nfeatures=700)

# Находим ключевые точки и описания с помощью #ORB.
kp1, desc1 = orb.detectAndCompute(img1, None)
kp2, desc2 = orb.detectAndCompute(img2, None)

# Находим совпадения ключевых точек с помощью Brute #Force Matcher.
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
matches = bf.match(desc1, desc2, None)

# Сортируем совпадения в порядке увеличения расстояния.
matches = sorted(matches, key=lambda x: x.distance)

# Отрисовываем лучшие совпадения.
img3 = cv.drawMatches(img1, kp1, img2, kp2,
                      matches[:MIN_NUM_KEYPOINT_MATCHES],
                      None)

cv.namedWindow('Matches', cv.WINDOW_NORMAL)
img3_resize = cv.resize(img3, (699, 700))
cv.imshow('Matches', img3_resize)
cv.waitKey(7000) # Keeps window open 7 seconds.
cv.destroyWindow('Matches')

# Оставляем только лучшие совпадения.
best_matches = matches[:MIN_NUM_KEYPOINT_MATCHES]

if len(best_matches) >= MIN_NUM_KEYPOINT_MATCHES:
    src_pts = np.zeros((len(best_matches), 2), dtype=np.float32)
    dst_pts = np.zeros((len(best_matches), 2), dtype=np.float32)

    for i, match in enumerate(best_matches):
        src_pts[i, :] = kp1[match.queryIdx].pt
        dst_pts[i, :] = kp2[match.trainIdx].pt

    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC)

    # Получаем размеры изображения 2.
    height, width = img2.shape
    img1_warped = cv.warpPerspective(img1, M, (width, height))

    cv.imwrite('montage_left_registered.JPG', img1_warped)
    cv.imwrite('montage_right_gray.JPG', img2)

else:
    print("\n{}\n".format('WARNING: Number of keypoint matches < 10!'))
```

practice_montage_difference_finder.py

practice_montage_difference_finder.py

```
import cv2 as cv

filename1 = 'montage_left.JPG'
filename2 = 'montage_right_gray.JPG'

img1 = cv.imread(filename1, cv.IMREAD_GRAYSCALE)
img2 = cv.imread(filename2, cv.IMREAD_GRAYSCALE)

# Абсолютная разница между изображениями 2 и 3:
diff_imgs1_2 = cv.absdiff(img1, img2)

cv.namedWindow('Difference', cv.WINDOW_NORMAL)
diff_imgs1_2_resize = cv.resize(diff_imgs1_2, (699, 700))
cv.imshow('Difference', diff_imgs1_2_resize)

crop_diff = diff_imgs1_2[10:2795, 10:2445] # x, y, w, h = 10, 10, 2790, 2440

# Применяем размытие для удаления лишнего шума.
blurred = cv.GaussianBlur(crop_diff, (5, 5), 0)

(minVal, maxVal, minLoc, maxLoc2) = cv.minMaxLoc(blurred)
cv.circle(img2, maxLoc2, 100, 0, 3)
x, y = int(img2.shape[1]/4), int(img2.shape[0]/4)
img2_resize = cv.resize(img2, (x, y))
cv.imshow('Change', img2_resize)
```

Глава 6. Победа в лунной гонке с помощью «Аполлона-8»

Симуляция шаблона поисков

practice_search_pattern.py

```
import time
import random
import turtle

SA_X = 600 # Ширина области поиска.
SA_Y = 480 # Длина области поиска.
TRACK_SPACING = 40 # Расстояние между треками поиска.

# Настройка экрана.
screen = turtle.Screen()
screen.setup(width=SA_X, height=SA_Y)
turtle.resizemode('user')
screen.title("Search Pattern")
rand_x = random.randint(0, int(SA_X / 2)) * random.choice([-1, 1])
rand_y = random.randint(0, int(SA_Y / 2)) * random.choice([-1, 1])

# Настройка изображений черепах  .
```

```
seaman_image = 'seaman.gif'
screen.addshape(seaman_image)
copter_image_left = 'helicopter_left.gif'
copter_image_right = 'helicopter_right.gif'
screen.addshape(copter_image_left)
screen.addshape(copter_image_right)

# Инстанцирование указателя моряка.
seaman = turtle.Turtle(seaman_image)
seaman.hideturtle()
seaman.penup()
seaman.setpos(rand_x, rand_y)
seaman.showturtle()

# Инстанцирование указателя вертолета.
turtle.shape(copter_image_right)
turtle.hideturtle()
turtle.pencolor('black')
turtle.penup()
turtle.setpos(-(int(SA_X / 2) - TRACK_SPACING), int(SA_Y / 2) - TRACK_
SPACING)
turtle.showturtle()
turtle.pendown()

# Выполняем шаблон поиска и объявляем о нахождении моряка.
for i in range(int(SA_Y / TRACK_SPACING)):
    turtle.fd(SA_X - TRACK_SPACING * 2)
    turtle.rt(90)
    turtle.fd(TRACK_SPACING / 2)
    turtle.rt(90)
    turtle.shape(copter_image_left)
    turtle.fd(SA_X - TRACK_SPACING * 2)
    turtle.lt(90)
    turtle.fd(TRACK_SPACING / 2)
    turtle.lt(90)
    turtle.shape(copter_image_right)
    if turtle.ycor() - seaman.ycor() <= 10:
        turtle.write("      Seaman found!",
                     align='left',
                     font=("Arial", 15, 'normal', 'bold', 'italic'))
        time.sleep(3)

        break
```

Заведи меня!

practice_grav_assist_stationary.py

```
"""gravity_assist_stationary.py

Луна приближается к неподвижному кораблю, закручивая его вокруг себя
и запуская вдаль.

Реализация: Эрик Т. Мортенсон
"""
```

```
from turtle import Shape, Screen, Turtle, Vec2D as Vec
import turtle
import math

# Пользовательский ввод:
G = 8 # Гравитационная константа, используемая в симуляции.
NUM_LOOPS = 4100 # Число временных шагов в симуляции.
Ro_X = 0 # Координата x стартовой позиции корабля.
Ro_Y = -50 # Координата y стартовой позиции корабля.
Vo_X = 0 # Компонент x скорости корабля.
Vo_Y = 0  # Компонент y скорости корабля.

MOON_MASS = 1_250_000


class GravSys():
    """Выполняем гравитационную симуляцию для n тел."""

    def __init__(self):
        self.bodies = []
        self.t = 0
        self.dt = 0.001

    def sim_loop(self):
        """Прогоняем тела из списка через временные шаги."""
        for _ in range(NUM_LOOPS):
            self.t += self.dt
            for body in self.bodies:
                body.step()


class Body(Turtle):
    """Небесный объект, вращающийся по орбите и проецирующий гравитационное
    поле."""
    def __init__(self, mass, start_loc, vel, gravsys, shape):
        super().__init__(shape=shape)
        self.gravsys = gravsys
        self.penup()
        self.mass=mass
        self.setpos(start_loc)
        self.vel = vel
        gravsys.bodies.append(self)
        self.pendown() # Раскомментируйте, чтобы рисовать путь позади
                         объектов


def acc(self):
    """Вычисляем совместное действие сил на тело и возвращаем компоненты
    вектора."""
    a = Vec(0,0)
    for body in self.gravsys.bodies:
        if body != self:
            r = body.pos() - self.pos()
            a += (G * body.mass / abs(r)**3) * r # единицы расстояние/время^2
    return a
```

```
    def step(self):
        """Вычисляем позицию, ориентацию и скорость тела."""
        dt = self.gravsys.dt
        a = self.acc()
        self.vel = self.vel + dt * a
        xOld, yOld = self.pos() # для ориентации корабля.
        self.setpos(self.pos() + dt * self.vel)
        xNew, yNew = self.pos() # для ориентации корабля.
        if self.gravsys.bodies.index(self) == 1: # the CSM
            dir_radians = math.atan2(yNew-yOld,xNew-xOld) # для ориентации
                                                            корабля.
            dir_degrees = dir_radians * 180 / math.pi # для ориентации
                                                        корабля.
            self.setheading(dir_degrees+90) # для ориентации корабля.

def main():
    # Настраиваем экран.
    screen = Screen()
    screen.setup(width=1.0, height=1.0) # для полноэкранного режима
    screen.bgcolor('black')
    screen.title("Gravity Assist Example")

    # Инстанцируем гравитационную систему.
    gravsys = GravSys()

    # Инстанцируем планету.
    image_moon = 'moon_27x27.gif'
    screen.register_shape(image_moon)
    moon = Body(MOON_MASS, (500, 0), Vec(-500, 0), gravsys, image_moon)
    moon.pencolor('gray')

    # Создаем командно-сервисный модуль (csm).
    csm = Shape('compound')
    cm = ((0, 30), (0, -30), (30, 0))
    csm.addcomponent(cm, 'red', 'red')
    sm = ((-60,30), (0, 30), (0, -30), (-60, -30))
    csm.addcomponent(sm, 'red', 'black')
    nozzle = ((-55, 0), (-90, 20), (-90, -20))
    csm.addcomponent(nozzle, 'red', 'red')
    screen.register_shape('csm', csm)

    # Инстанцируем указатель CSM "Аполлона-8"
    ship = Body(1, (Ro_X, Ro_Y), Vec(Vo_X, Vo_Y), gravsys, "csm")
    ship.shapesize(0.2)
    ship.color('red') # цвет траектории
    ship.getscreen().tracer(1, 0)
    ship.setheading(90)

    gravsys.sim_loop()

if __name__=='__main__':
    main()
```

Останови меня!

practice_grav_assist_intersecting.py

```
"""gravity_assist_intersecting.py

Орбиты Луны и корабля пересекаются, в результате чего Луна замедляется
и разворачивает корабль.

Реализация: Эрик Т. Мортенсон
"""
from turtle import Shape, Screen, Turtle, Vec2D as Vec
import turtle
import math
import sys

# Пользовательский ввод:
G = 8 # Гравитационная константа, используемая в симуляции.
NUM_LOOPS = 7000 # Число временных шагов в симуляции.
Ro_X = -152.18 # Координата x стартовой позиции корабля.
Ro_Y = 329.87 # Координата y стартовой позиции корабля.
Vo_X = 423.10 # Компонент x скорости выхода на транслунную орбиту.
Vo_Y = -512.26 # Компонент y скорости выхода на транслунную орбиту.

MOON_MASS = 1_250_000


class GravSys():
    """Выполняем гравитационную симуляцию для n тел."""

    def __init__(self):
        self.bodies = []
        self.t = 0
        self.dt = 0.001

    def sim_loop(self):
        """Прогоняем тела из списка по временным шагам."""
        for index in range(NUM_LOOPS): # остановка моделирования через
                                         некоторое время
            self.t += self.dt
            for body in self.bodies:
                body.step()


class Body(Turtle):
    """Небесный объект, вращающийся по орбите и проецирующий гравитационное
    поле."""
    def __init__(self, mass, start_loc, vel, gravsys, shape):
        super().__init__(shape=shape)
        self.gravsys = gravsys
        self.penup()
        self.mass=mass
        self.setpos(start_loc)
        self.vel = vel
        gravsys.bodies.append(self)
        self.pendown() # Раскомментируйте, чтобы рисовать позади указателя
                         его путь.
```

```
    def acc(self):
        """Вычисляем комбинированную силу, действующую на тело, и возвращаем
        компоненты вектора."""
        a = Vec(0,0)
        for body in self.gravsys.bodies:
            if body != self:
                r = body.pos() - self.pos()
                a += (G * body.mass / abs(r)**3) * r # единицы расстояние/
                                                       время^2
        return a

    def step(self):
        """Вычисляем позицию, ориентацию и скорость тела."""
        dt = self.gravsys.dt
        a = self.acc()
        self.vel = self.vel + dt * a
        xOld, yOld = self.pos() # для ориентации корабля
        self.setpos(self.pos() + dt * self.vel)
        xNew, yNew = self.pos() # для ориентации корабля
        if self.gravsys.bodies.index(self) == 1: # CSM
            dir_radians = math.atan2(yNew-yOld,xNew-xOld) # для ориентации
                                                            корабля
            dir_degrees = dir_radians * 180 / math.pi # для ориентации
                                                        корабля
            self.setheading(dir_degrees+90) # для ориентации корабля


def main():
    # Настройка экрана
    screen = Screen()
    screen.setup(width=1.0, height=1.0) # для полноэкранного режима
    screen.bgcolor('black')
    screen.title("Gravity Assist Example")

    # Инстанцируем гравитационную систему
    gravsys = GravSys()

    # Инстанцируем планету
    image_moon = 'moon_27x27.gif'
    screen.register_shape(image_moon)
    moon = Body(MOON_MASS, (-250, 0), Vec(500, 0), gravsys, image_moon)
    moon.pencolor('gray')

    # Создаем фигуру командно-сервисного модуля (csm)
    csm = Shape('compound')
    cm = ((0, 30), (0, -30), (30, 0))
    csm.addcomponent(cm, 'red', 'red')
    sm = ((-60,30), (0, 30), (0, -30), (-60, -30))
    csm.addcomponent(sm, 'red', 'black')
    nozzle = ((-55, 0), (-90, 20), (-90, -20))
    csm.addcomponent(nozzle, 'red', 'red')
    screen.register_shape('csm', csm)

    # Инстанцируем указатель CSM "Аполлона-8"
    ship = Body(1, (Ro_X, Ro_Y), Vec(Vo_X, Vo_Y), gravsys, "csm")
```

```
    ship.shapesize(0.2)
    ship.color('red') # цвет траектории
    ship.getscreen().tracer(1, 0)
    ship.setheading(90)
    gravsys.sim_loop()

if __name__=='__main__':
    main()
```

Глава 7. Выбор мест высадки на Марсе

Убеждаемся, что рисунки становятся частью изображения

practice_confirm_drawing_part_of_image.py

```
"""Убеждаемся, что рисунки становятся частью изображения в OpenCV."""
import numpy as np
import cv2 as cv

IMG = cv.imread('mola_1024x501.png', cv.IMREAD_GRAYSCALE)

ul_x, ul_y = 0, 167
lr_x, lr_y = 32, 183
rect_img = IMG[ul_y : lr_y, ul_x : lr_x]

def run_stats(image):
    """Вычисляем статистики для массива numpy, полученного из изображения."""
    print('mean = {}'.format(np.mean(image)))
    print('std = {}'.format(np.std(image)))
    print('ptp = {}'.format(np.ptp(image)))
    print()
    cv.imshow('img', IMG)
    cv.waitKey(1000)

# Статистики без рисунка на экране:
print("No drawing")
run_stats(rect_img)

# Статистики для прямоугольника с белой рамкой:
print("White outlined rectangle")
cv.rectangle(IMG, (ul_x, ul_y), (lr_x, lr_y), (255, 0, 0), 1)
run_stats(rect_img)

# Статистики для прямоугольника, закрашенного белым:
print("White-filled rectangle")
cv.rectangle(IMG, (ul_x, ul_y), (lr_x, lr_y), (255, 0, 0), -1)
run_stats(rect_img)
```

Визуализация профиля высоты

practice_profile_olympus.py

```
"""Профиль высоты горы Олимп с запада на восток."""
from PIL import Image, ImageDraw
from matplotlib import pyplot as plt
```

```
# Загружаем изображение и получаем значения x и z вдоль горизонтального
  профиля, параллельного y координате.
y_coord = 202
im = Image.open('mola_1024x512_200mp.jpg').convert('L')
width, height = im.size
x_vals = [x for x in range(width)]
z_vals = [im.getpixel((x, y_coord)) for x in x_vals]

# Рисуем профиль на изображении MOLA.
draw = ImageDraw.Draw(im)
draw.line((0, y_coord, width, y_coord), fill=255, width=3)
draw.text((100, 165), 'Olympus Mons', fill=255)
im.show()

# Строим графическое представление профиля.
fig, ax = plt.subplots(figsize=(9, 4))
axes = plt.gca()
axes.set_ylim(0, 400)
ax.plot(x_vals, z_vals, color='black')
ax.set(xlabel='x-coordinate',
       ylabel='Intensity (height)',
       title="Mars Elevation Profile (y = 202)")
ratio = 0.15 # Уменьшаем вертикальное преувеличение в профиле рельефа.
xleft, xright = ax.get_xlim()
ybase, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybase-ytop)) * ratio)
plt.text(0, 310, 'WEST', fontsize=10)
plt.text(980, 310, 'EAST', fontsize=10)
plt.text(100, 280, 'Olympus Mons', fontsize=8)
##ax.grid()
plt.show()
```

Отображение в 3D

practice_3d_plotting.py

```
"""Создаем графическое представление карты Марса MOLA в 3D.
Реализация: Эрик Т. Мортенсон."""
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

IMG_GRAY = cv.imread('mola_1024x512_200mp.jpg', cv.IMREAD_GRAYSCALE)

x = np.linspace(1023, 0, 1024)
y = np.linspace(0, 511, 512)

X, Y = np.meshgrid(x, y)
Z = IMG_GRAY[0:512, 0:1024]

fig = plt.figure()
ax = plt.axes(projection='3d')
```

```
ax.contour3D(X, Y, Z, 150, cmap='gist_earth') # 150=количество контуров
ax.auto_scale_xyz([1023, 0], [0, 511], [0, 500])
plt.show()
```

Совмещение карт

В этом практическом проекте используются две программы, practice_geo_map_step_1of2.py и practice_geo_map_step_2of2.py, которые нужно выполнять по порядку.

practice_geo_map_step_1of2.py

practice_geo_map_step_1of2.py

```
"""Применяем заданный в значениях пикселей порог к полутоновому изображению
и сохраняем результат в файл."""
import cv2 as cv

IMG_GEO = cv.imread('Mars_Global_Geology_Mariner9_1024.jpg',
                     cv.IMREAD_GRAYSCALE)
cv.imshow('map', IMG_GEO)
cv.waitKey(1000)
img_copy = IMG_GEO.copy()
lower_limit = 170 # Минимальное полутоновое значение для вулканических
                    отложений
upper_limit = 185 # Максимальное полутоновое значение для вулканических
                    отложений

# Используем изображение 1024x512
for x in range(1024):
    for y in range(512):
        if lower_limit <= img_copy[y, x] <= upper_limit:
            img_copy[y, x] = 1 # Устанавливаем на 255, чтобы визуализировать
                                 результат.
        else:
            img_copy[y, x] = 0

cv.imwrite('geo_thresh.jpg', img_copy)
cv.imshow('thresh', img_copy)
cv.waitKey(0)
```

practice_geo_map_step_2of2.py

practice_geo_map_step_2of2.py

```
"""Выбираем места высадки на Марсе, опираясь на гладкость поверхности
и геологические особенности."""
import tkinter as tk
from PIL import Image, ImageTk
import numpy as np
import cv2 as cv
```

```
# Константы: пользовательский ввод:
IMG_GRAY = cv.imread('mola_1024x512_200mp.jpg', cv.IMREAD_GRAYSCALE)
IMG_GEO = cv.imread('geo_thresh.jpg', cv.IMREAD_GRAYSCALE)
IMG_COLOR = cv.imread('mola_color_1024x506.png')
RECT_WIDTH_KM = 670 # Ширина прямоугольного участка местности в километрах.
RECT_HT_KM = 335 # Высота прямоугольного участка местности в километрах.
MIN_ELEV_LIMIT = 60 # Значения интенсивности (0-255).
MAX_ELEV_LIMIT = 255
NUM_CANDIDATES = 20 # Количество предположительных мест высадки для
отображения.

#-----------------------------------------------------------------------------
# Константы: производные и фиксированные:
IMG_GRAY_GEO = IMG_GRAY * IMG_GEO
IMG_HT, IMG_WIDTH = IMG_GRAY.shape
MARS_CIRCUM = 21344 # Окружность в километрах.
PIXELS_PER_KM = IMG_WIDTH / MARS_CIRCUM
RECT_WIDTH = int(PIXELS_PER_KM * RECT_WIDTH_KM)
RECT_HT = int(PIXELS_PER_KM * RECT_HT_KM)
LAT_30_N = int(IMG_HT / 3)
LAT_30_S = LAT_30_N * 2
STEP_X = int(RECT_WIDTH / 2) # Деление на 4 дает больше прямоугольников.
STEP_Y = int(RECT_HT / 2) # Деление на 4 дает больше прямоугольников.

# Создаем объекты экран tkinter и canvas для рисования.
screen = tk.Tk()
canvas = tk.Canvas(screen, width=IMG_WIDTH, height=IMG_HT + 130)


class Search():
    """Считываем изображение и определяем места посадки на основе вводных
    критериев."""

    def __init__(self, name):
        self.name = name
        self.rect_coords = {}
        self.rect_means = {}
        self.rect_ptps = {}
        self.rect_stds = {}
        self.ptp_filtered = []
        self.std_filtered = []
        self.high_graded_rects = []


    def run_rect_stats(self):
        """Определяем прямоугольные области поиска и вычисляем их внутренние
        статистики."""
        ul_x, ul_y = 0, LAT_30_N
        lr_x, lr_y = RECT_WIDTH, LAT_30_N + RECT_HT
        rect_num = 1

        while True:
            rect_img = IMG_GRAY_GEO[ul_y : lr_y, ul_x : lr_x]
            self.rect_coords[rect_num] = [ul_x, ul_y, lr_x, lr_y]
```

```
            if MAX_ELEV_LIMIT >= np.mean(rect_img) >= MIN_ELEV_LIMIT:
                self.rect_means[rect_num] = np.mean(rect_img)
                self.rect_ptps[rect_num] = np.ptp(rect_img)
                self.rect_stds[rect_num] = np.std(rect_img)
            rect_num += 1

            # Смещаем прямоугольник.
            ul_x += STEP_X
            lr_x = ul_x + RECT_WIDTH if
            if lr_x > IMG_WIDTH:
              ul_x = 0
                ul_y += STEP_Y
                lr_x = RECT_WIDTH
                lr_y += STEP_Y
            if lr_y > LAT_30_S + STEP_Y:
                break

    def draw_qc_rects(self):
        """Рисуем на изображении накладывающиеся прямоугольники поиска
        для проверки."""
        img_copy = IMG_GRAY_GEO.copy()
        rects_sorted = sorted(self.rect_coords.items(), key=lambda x: x[0])
        print("\nRect Number and Corner Coordinates (ul_x, ul_y, lr_x,
              lr_y):")
        for k, v in rects_sorted:
            print("rect: {}, coords: {}".format(k, v))
            cv.rectangle(img_copy,
                         (self.rect_coords[k][0], self.rect_coords[k][1]),
                         (self.rect_coords[k][2], self.rect_coords[k][3]),
                         (255, 0, 0), 1)
        cv.imshow('QC Rects {}'.format(self.name), img_copy)
        cv.waitKey(3000)
        cv.destroyAllWindows()

    def sort_stats(self):
        """Сортировка словарей по значениям и создание списков из N лучших
        ключей."""
        ptp_sorted = (sorted(self.rect_ptps.items(), key=lambda x: x[1]))
        self.ptp_filtered = [x[0] for x in ptp_sorted[:NUM_CANDIDATES]]
        std_sorted = (sorted(self.rect_stds.items(), key=lambda x: x[1]))
        self.std_filtered = [x[0] for x in std_sorted[:NUM_CANDIDATES]]

        # Создаем список прямоугольников, где отфильтрованные std
          и ptp совпадают.
        for rect in self.std_filtered:
            if rect in self.ptp_filtered:
                self.high_graded_rects.append(rect)

    def draw_filtered_rects(self, image, filtered_rect_list):
        """Отрисовываем прямоугольники из списка на изображении и возвращаем
        его."""
        img_copy = image.copy()
        for k in filtered_rect_list:
            cv.rectangle(img_copy,
```

```
                    (self.rect_coords[k][0], self.rect_coords[k][1]),
                    (self.rect_coords[k][2], self.rect_coords[k][3]),
                    (255, 0, 0), 1)
            cv.putText(img_copy, str(k),
                       (self.rect_coords[k][0] + 1,
                        self.rect_coords[k][3]- 1),
                       cv.FONT_HERSHEY_PLAIN, 0.65, (255, 0, 0), 1)

        # Рисуем границы широты.
        cv.putText(img_copy, '30 N', (10, LAT_30_N - 7),
                   cv.FONT_HERSHEY_PLAIN, 1, 255)
        cv.line(img_copy, (0, LAT_30_N), (IMG_WIDTH, LAT_30_N),
                (255, 0, 0), 1)
        cv.line(img_copy, (0, LAT_30_S), (IMG_WIDTH, LAT_30_S),
                (255, 0, 0), 1)
        cv.putText(img_copy, '30 S', (10, LAT_30_S + 16),
                   cv.FONT_HERSHEY_PLAIN, 1, 255)

        return img_copy

    def make_final_display(self):
        """Используем Tk для показа карты итоговых прямоугольников и вывода
        их статистик."""
        screen.title('Sites by MOLA Gray STD & PTP {}
                      Rect'.format(self.name))
        # Рисуем высокоточные прямоугольники на цветной карте высот.
        img_color_rects = self.draw_filtered_rects(IMG_COLOR,
                                                   self.high_graded_rects)
        # Преобразуем изображение из CV BGR в RGB для использования
          с помощью Tkinter.
        img_converted = cv.cvtColor(img_color_rects, cv.COLOR_BGR2RGB)
        img_converted = ImageTk.PhotoImage(Image.fromarray(img_converted))
        canvas.create_image(0, 0, image=img_converted, anchor=tk.NW)
        # Добавляем в нижней части canvas статистики для каждого
          прямоугольника.
        txt_x = 5
        txt_y = IMG_HT + 15
        for k in self.high_graded_rects:
            canvas.create_text(txt_x, txt_y, anchor='w', font=None,
                               text=
                               "rect={} mean elev={:.1f} std={:.2f} ptp={}"
                               .format(k, self.rect_means[k],
                                       self.rect_stds[k],
                                       self.rect_ptps[k]))
            txt_y += 15
            if txt_y >= int(canvas.cget('height')) - 10:
                txt_x += 300
                txt_y = IMG_HT + 15
        canvas.pack()
        screen.mainloop()

def main():
    app = Search('670x335 km')
    app.run_rect_stats()
```

```
    app.draw_qc_rects()
    app.sort_stats()
    ptp_img = app.draw_filtered_rects(IMG_GRAY_GEO, app.ptp_filtered)
    std_img = app.draw_filtered_rects(IMG_GRAY_GEO, app.std_filtered)

    # Отображаем отфильтрованные прямоугольники на полутоновой карте.
    cv.imshow('Sorted by ptp for {} rect'.format(app.name), ptp_img)
    cv.waitKey(3000)
    cv.imshow('Sorted by std for {} rect'.format(app.name), std_img)
    cv.waitKey(3000)

    app.make_final_display() # Включает в себя вызов mainloop()

if __name__ == '__main__':
    main()
```

Глава 8. Обнаружение далеких экзопланет

Обнаружение инопланетных мегаструктур

practice_tabbys_star.py

```
"""Симулируем транзит инопланетного массива на фоне звезды и отображаем
кривую блеска."""
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

IMG_HT = 400
IMG_WIDTH = 500
BLACK_IMG = np.zeros((IMG_HT, IMG_WIDTH), dtype='uint8')
STAR_RADIUS = 165
EXO_START_X = -250
EXO_START_Y = 150
EXO_DX = 3
NUM_FRAMES = 500

def main():
    intensity_samples = record_transit(EXO_START_X, EXO_START_Y)
    rel_brightness = calc_rel_brightness(intensity_samples)
    plot_light_curve(rel_brightness)

def record_transit(exo_x, exo_y):
    """Рисуем проходящий на фоне звезды массив и возвращаем список изменений
    в интенсивности ее свечения."""
    intensity_samples = []
    for _ in range(NUM_FRAMES):
        temp_img = BLACK_IMG.copy()
        # Рисуем звезду:
        cv.circle(temp_img, (int(IMG_WIDTH / 2), int(IMG_HT / 2)),
                  STAR_RADIUS, 255, -1)
        # Рисуем инопланетный массив:
```

```
        cv.rectangle(temp_img, (exo_x, exo_y),
                     (exo_x + 20, exo_y + 140), 0, -1)
        cv.rectangle(temp_img, (exo_x - 360, exo_y),
                     (exo_x + 10, exo_y + 140), 0, 5)
        cv.rectangle(temp_img, (exo_x - 380, exo_y),
                     (exo_x - 310, exo_y + 140), 0, -1)
        intensity = temp_img.mean()
        cv.putText(temp_img, 'Mean Intensity = {}'.format(intensity),
                   (5, 390),
                   cv.FONT_HERSHEY_PLAIN, 1, 255)
        cv.imshow('Transit', temp_img)
        cv.waitKey(10)
        intensity_samples.append(intensity)
        exo_x += EXO_DX
    return intensity_samples

def calc_rel_brightness(intensity_samples):
    """Возвращаем из списка значений интенсивности список значений
    относительной яркости."""
    rel_brightness = []
    max_brightness = max(intensity_samples)
    for intensity in intensity_samples:
        rel_brightness.append(intensity / max_brightness)
    return rel_brightness

def plot_light_curve(rel_brightness):
    """Выводим график изменений относительной яркости во времени."""
    plt.plot(rel_brightness, color='red', linestyle='dashed',
             linewidth=2)
    plt.title('Relative Brightness vs. Time')
    plt.xlim(-150, 500)
    plt.show()

if __name__ == '__main__':
    main()
```

Обнаружение транзита астероидов

practice_asteroids.py

```
"""Симулируем транзит астероидов и рисуем график кривой блеска."""
import random
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

STAR_RADIUS = 165
BLACK_IMG = np.zeros((400, 500, 1), dtype="uint8")
NUM_ASTEROIDS = 15
NUM_LOOPS = 170


class Asteroid():
    """Рисуем на изображении круг, обозначающий астероид."""
```

```
    def __init__(self, number):
        self.radius = random.choice((1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
                                     2, 3))
        self.x = random.randint(-30, 60)
        self.y = random.randint(220, 230)
        self.dx = 3

    def move_asteroid(self, image):
        """Рисуем и перемещаем объект астероида."""
        cv.circle(image, (self.x, self.y), self.radius, 0, -1)
        self.x += self.dx


def record_transit(start_image):
    """Симулируем транзит астероидов на фоне звезды и возвращаем список
    значений интенсивности."""
    asteroid_list = []
    intensity_samples = []

    for i in range(NUM_ASTEROIDS):
        asteroid_list.append(Asteroid(i))

    for _ in range(NUM_LOOPS):
        temp_img = start_image.copy()
        # Рисуем звезду.
        cv.circle(temp_img, (250, 200), STAR_RADIUS, 255, -1)
        for ast in asteroid_list:
            ast.move_asteroid(temp_img)
        intensity = temp_img.mean()
        cv.putText(temp_img, 'Mean Intensity = {}'.format(intensity),
                   (5, 390), cv.FONT_HERSHEY_PLAIN, 1, 255)
        cv.imshow('Transit', temp_img)
        intensity_samples.append(intensity)
        cv.waitKey(50)
    cv.destroyAllWindows()
    return intensity_samples

def calc_rel_brightness(image):
    """Вычисляем величины относительной яркости и возвращаем их список."""
    rel_brightness = record_transit(image)
    max_brightness = max(rel_brightness)
    for i, j in enumerate(rel_brightness):
        rel_brightness[i] = j / max_brightness
    return rel_brightness


def plot_light_curve(rel_brightness):
    "Рисуем график кривой блеска на основе списка значений относительной
    яркости."""
    plt.plot(rel_brightness, color='red', linestyle='dashed',
             linewidth=2, label='Relative Brightness')
    plt.legend(loc='upper center')
    plt.title('Relative Brightness vs. Time')
    plt.show()
```

```
relative_brightness = calc_rel_brightness(BLACK_IMG)
plot_light_curve(relative_brightness)
```

Добавление эффекта потемнения к краю

practice_limb_darkening.py

```
"""Симулируем транзит экзопланеты, рисуем график кривой ее блеска и оцениваем
радиус."""
import cv2 as cv
import matplotlib.pyplot as plt

IMG_HT = 400
IMG_WIDTH = 500
BLACK_IMG = cv.imread('limb_darkening.png', cv.IMREAD_GRAYSCALE)
EXO_RADIUS = 7
EXO_START_X = 40
EXO_START_Y = 230
EXO_DX = 3
NUM_FRAMES = 145

def main():
    intensity_samples = record_transit(EXO_START_X, EXO_START_Y)
    relative_brightness = calc_rel_brightness(intensity_samples)
    plot_light_curve(relative_brightness)

def record_transit(exo_x, exo_y):
    """Рисуем планету, проходящую на фоне звезды, и возвращаем список
    со значениями изменяющейся интенсивности."""
    intensity_samples = []
    for _ in range(NUM_FRAMES):
        temp_img = BLACK_IMG.copy()
        # Рисуем экзопланету:
        cv.circle(temp_img, (exo_x, exo_y), EXO_RADIUS, 0, -1)
        intensity = temp_img.mean()
        cv.putText(temp_img, 'Mean Intensity = {}'.format(intensity),
                   (5, 390),
                   cv.FONT_HERSHEY_PLAIN, 1, 255)
        cv.imshow('Transit', temp_img)
        cv.waitKey(30)
        intensity_samples.append(intensity)
        exo_x += EXO_DX
    return intensity_samples

def calc_rel_brightness(intensity_samples):
    """Возвращаем из списка интенсивности список значений относительной
    яркости."""
    rel_brightness = []
    max_brightness = max(intensity_samples)
    for intensity in intensity_samples:
        rel_brightness.append(intensity / max_brightness)
    return rel_brightness
```

```
def plot_light_curve(rel_brightness):
    """Рисуем график изменений относительной яркости во времени."""
    plt.plot(rel_brightness, color='red', linestyle='dashed',
             linewidth=2, label='Relative Brightness')
    plt.legend(loc='upper center')
    plt.title('Relative Brightness vs. Time')
##    plt.ylim(0.995, 1.001)
    plt.show()

if __name__ == '__main__':
    main()
```

Обнаружение инопланетной армады

practice_alien_armada.py

```
"""Симулируем транзит армады инопланетных кораблей и отображаем кривую
блеска."""
import random
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

STAR_RADIUS = 165
BLACK_IMG = np.zeros((400, 500, 1), dtype="uint8")
NUM_SHIPS = 5
NUM_LOOPS = 300 # Количество выполняемых шагов симуляции.

class Ship():
    """Рисуем и перемещаем объект корабля на изображении."""

    def __init__(self, number):
        self.number = number
        self.shape = random.choice(['>>>|==H[X)',
                                    '>>|==H[XX}=))-',
                                    '>>|==H[XX]=(-'])
        self.size = random.choice([0.7, 0.8, 1])
        self.x = random.randint(-180, -80)
        self.y = random.randint(80, 350)
        self.dx = random.randint(2, 4)

    def move_ship(self, image):
        """Рисуем и перемещаем объект корабля."""
        font = cv.FONT_HERSHEY_PLAIN
        cv.putText(img=image,
                   text=self.shape,
                   org=(self.x, self.y),
                   fontFace=font,
                   fontScale=self.size,
                   color=0,
                   thickness=5)
        self.x += self.dx
```

```
def record_transit(start_image):
    """Выполняем симуляцию и возвращаем список измерений интенсивности
    для каждого шага симуляции."""
    ship_list = []
    intensity_samples = []

    for i in range(NUM_SHIPS):
        ship_list.append(Ship(i))

    for _ in range(NUM_LOOPS):
        temp_img = start_image.copy()
        cv.circle(temp_img, (250, 200), STAR_RADIUS, 255, -1) # Звезда.
        for ship in ship_list:
            ship.move_ship(temp_img)
        intensity = temp_img.mean()
        cv.putText(temp_img, 'Mean Intensity = {}'.format(intensity),
                   (5, 390), cv.FONT_HERSHEY_PLAIN, 1, 255)
        cv.imshow('Transit', temp_img)
        intensity_samples.append(intensity)
        cv.waitKey(50)
    cv.destroyAllWindows()
    return intensity_samples

def calc_rel_brightness(image):
    """Возвращаем список измерений относительной яркости для планетарного
    транзита."""
    rel_brightness = record_transit(image)
    max_brightness = max(rel_brightness)
    for i, j in enumerate(rel_brightness):
        rel_brightness[i] = j / max_brightness
    return rel_brightness

def plot_light_curve(rel_brightness):
    """Рисуем график кривой относительной яркости во времени."""
    plt.plot(rel_brightness, color='red', linestyle='dashed',
             linewidth=2, label='Relative Brightness')
    plt.legend(loc='upper center')
    plt.title('Relative Brightness vs. Time')
    plt.show()

relative_brightness = calc_rel_brightness(BLACK_IMG)
plot_light_curve(relative_brightness)
```

Обнаружение планеты с Луной

practice_planet_moon.py

```
"""Анимация Луны реализована Эриком Т. Мортенсоном."""
import math
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
```

```
IMG_HT = 500
IMG_WIDTH = 500
BLACK_IMG = np.zeros((IMG_HT, IMG_WIDTH, 1), dtype='uint8')
STAR_RADIUS = 200
EXO_RADIUS = 20
EXO_START_X = 20
EXO_START_Y = 250
MOON_RADIUS = 5
NUM_DAYS = 200 # Количество дней в году.

def main():
    intensity_samples = record_transit(EXO_START_X, EXO_START_Y)
    relative_brightness = calc_rel_brightness(intensity_samples)
    print('\nestimated exoplanet radius = {:.2f}\n'
          .format(STAR_RADIUS * math.sqrt(max(relative_brightness)
                                          -min(relative_brightness))))
    plot_light_curve(relative_brightness)

def record_transit(exo_x, exo_y):
    """Рисуем проходящую на фоне звезды планету и возвращаем список значений
    изменения интенсивности."""
    intensity_samples = []
    for dt in range(NUM_DAYS):
        temp_img = BLACK_IMG.copy()
        # Рисуем звезду:
        cv.circle(temp_img, (int(IMG_WIDTH / 2), int(IMG_HT/2)),
                  STAR_RADIUS, 255, -1)
        # Рисуем экзопланету.
        cv.circle(temp_img, (int(exo_x), int(exo_y)), EXO_RADIUS, 0, -1)
        # Рисуем луну.
        if dt != 0:
            cv.circle(temp_img, (int(moon_x), int(moon_y)), MOON_RADIUS,
                      0, -1)
        intensity = temp_img.mean()
        cv.putText(temp_img, 'Mean Intensity = {}'.format(intensity),
                   (5, 10),
                   cv.FONT_HERSHEY_PLAIN, 1, 255)
        cv.imshow('Transit', temp_img)
        cv.waitKey(10)
        intensity_samples.append(intensity)
        exo_x = IMG_WIDTH / 2 - (IMG_WIDTH / 2 - 20) * \
                math.cos(2 * math.pi * dt / (NUM_DAYS)*(1 / 2))
        moon_x = exo_x + \
                 3 * EXO_RADIUS * math.sin(2 * math.pi * dt / NUM_DAYS *(5))
        moon_y = IMG_HT / 2 - \
                 0.25 * EXO_RADIUS * \
                 math.sin(2 * math.pi * dt / NUM_DAYS * (5))
    cv.destroyAllWindows()

    return intensity_samples
```

```
def calc_rel_brightness(intensity_samples):
    """Возвращаем из списка интенсивности список значений относительной
    яркости."""
    rel_brightness = []
    max_brightness = max(intensity_samples)
    for intensity in intensity_samples:
        rel_brightness.append(intensity / max_brightness)
    return rel_brightness

def plot_light_curve(rel_brightness):
    """Рисуем график изменений относительной яркости во времени."""
    plt.plot(rel_brightness, color='red', linestyle='dashed',
             linewidth=2, label='Relative Brightness')
    plt.legend(loc='upper center')
    plt.title('Relative Brightness vs. Time')
    plt.show()

if __name__ == '__main__':
    main()
```

Рисунок А.1 обобщает вывод программы practice_planet_moon.py.

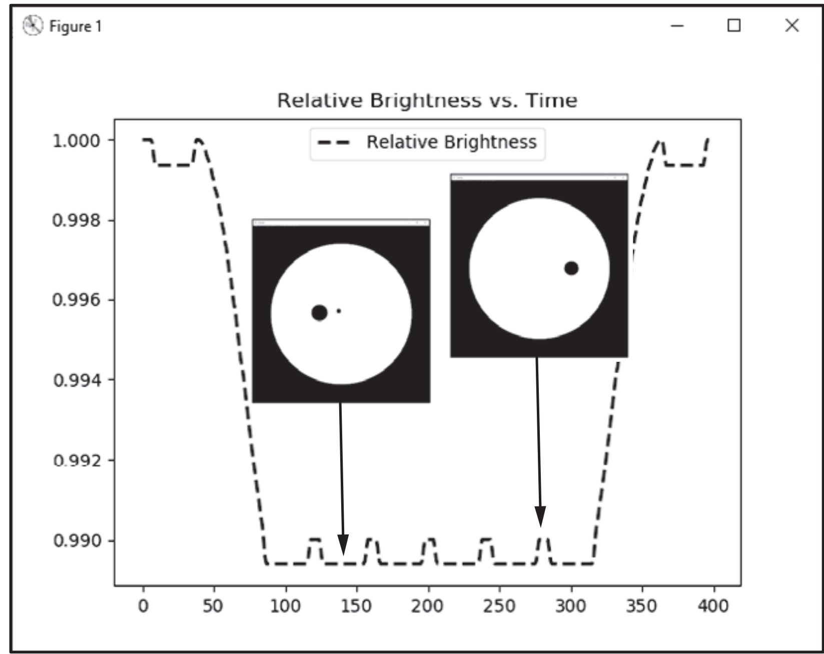


Рис. А.1. Кривая блеска для планеты и Луны, когда Луна проходит позади планеты

Измерение продолжительности экзопланетного дня

practice_length_of_day.py

```
"""Считываем изображения, вычисляем среднюю интенсивность, рисуем график
изменения относительной интенсивности во времени."""
import os
from statistics import mean
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal # См. главу 1 для установки scipy.

# Переключаемся на каталог с изображениями.
os.chdir('br549_pixelated')
images = sorted(os.listdir())
intensity_samples = []

# Конвертируем изображения в полутоновые и создаем список средних значений
  интенсивности.
for image in images:
    img = cv.imread(image, cv.IMREAD_GRAYSCALE)
    intensity = img.mean()
    intensity_samples.append(intensity)

# Генерируем список значений относительной интенсивности.
    rel_intensity = intensity_samples[:]
    max_intensity = max(rel_intensity)
    for i, j in enumerate(rel_intensity):
        rel_intensity[i] = j / max_intensity

# Рисуем график значений интенсивности относительно номера шага симуляции
  (альтернатива времени).
plt.plot(rel_intensity, color='red', marker='o', linestyle='solid',
         linewidth=2, markersize=0, label='Relative Intensity')
plt.legend(loc='upper center')
plt.title('Exoplanet BR549 Relative Intensity vs. Time')
plt.ylim(0.8, 1.1)
plt.xticks(np.arange(0, 50, 5))
plt.grid()
print("\nManually close plot window after examining to continue program.")
plt.show()

# Находим период/продолжительность суток.
# Оцениваем максимальную высоту и границы разделения (distance) на графике.
# Параметры height и distance представляют значения >= границ.
peaks = signal.find_peaks(rel_intensity, height=0.95, distance=5)
print(f"peaks = {peaks}")
print("Period = {}".format(mean(np.diff(peaks[0]))))
```

Глава 9. Как различить своих и чужих

Размытие лиц

practice_blur.py

```
import cv2 as cv

path = "C:/Python372/Lib/site-packages/cv2/data/"
face_cascade = cv.CascadeClassifier(path + 'haarcascade_frontalface_alt.xml')

cap = cv.VideoCapture(0)

while True:
    _, frame = cap.read()
    face_rects = face_cascade.detectMultiScale(frame, scaleFactor=1.2,
                                               minNeighbors=3)

    for (x, y, w, h) in face_rects:
        face = cv.blur(frame[y:y + h, x:x + w], (25, 25))
        frame[y:y + h, x: x + w] = face
        cv.rectangle(frame, (x,y), (x+w, y+h), (0, 255, 0), 2)

    cv.imshow('frame', frame)
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv.destroyAllWindows()
```

Глава 10. Ограничение доступа по принципу распознавания лиц

Усложняем проект: добавление пароля и видеозахвата

Следующий сниппет реализует часть проекта, связанную с распознаванием лиц из видеопотока.

challenge_video_recognize.py

```
"""Распознаем лицо капитана Демминга в видеокадре."""
import cv2 as cv

names = {1: "Demming"}

# Устанавливаем путь к каскадам Хаара
path = "C:/Python372/Lib/site-packages/cv2/data/"
detector = cv.CascadeClassifier(path + 'haarcascade_frontalface_default.xml')

# Устанавливаем распознаватель лиц и загружаем обученные данные.
recognizer = cv.face.LBPHFaceRecognizer_create()
recognizer.read('lbph_trainer.yml')

# Подготавливаем камеру.
```

```
cap = cv.VideoCapture(0)
if not cap.isOpened():
    print("Could not open video device.")
##cap.set(3, 320) # Ширина кадра.
##cap.set(4, 240) # Высота кадра.

while True:
    _, frame = cap.read()
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    face_rects = detector.detectMultiScale(gray,
                                           scaleFactor=1.2,
                                           minNeighbors=5)

    for (x, y, w, h) in face_rects:
        # Изменяем размер входного изображения, приближая его к размеру
          обучающего изображения.
        gray_resize = cv.resize(gray[y:y + h, x:x + w],
                                (100, 100),
                                cv.INTER_LINEAR)
        predicted_id, dist = recognizer.predict(gray_resize)
        if predicted_id == 1 and dist <= 110:
            name = names[predicted_id]
        else:
            name = 'unknown'
        cv.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 0), 2)
        cv.putText(frame, name, (x + 1, y + h -5),
                   cv.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 1)
        cv.imshow('frame', frame)

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

cap.release()
cv.destroyAllWindows()
```