Нижегородский государственный университет им. Н.И. Лобачевского
Факультет вычислительной математики и кибернетики

Учебный курс «Разработка мультимедийных приложений с использованием библиотек OpenCV и IPP»

Лекция
Начало работы с библиотекой OpenCV

Писаревский В.Н.

При поддержке компании Intel

Нижний Новгород
2013

Содержание

1. НАЧАЛО РАБОТЫ С БИБЛИОТЕКОЙ OPENCV 3

1.1. ОБЗОР БИБЛИОТЕКИ 3
1.2. OPENCV НА PYTHON'Е 6
1.2.1. Первая программа 7
1.2.2. Вторая программа: добавляем видео 8
1.2.3. Третья программа: добавляем детектирование лиц 9
1.2.4. Несколько замечаний об использовании связки Python + OpenCV 11
1.3. OPENCV НА C/C++ 12
1.3.1. Первая OpenCV-программа на C++ 12
1.3.2. Работа с матрицами 13
1.3.3. Пример: поиск плоских объектов 16

2. ЛИТЕРАТУРА 20

1. Начало работы с библиотекой OpenCV

Цель настоящего раздела – создать правильное впечатление о возможностях библиотеки OpenCV (http://opencv.org) и дать начальный толчок для дальнейшего его изучения.

1.1. Обзор библиотеки

OpenCV – de facto самая популярная библиотека компьютерного зрения. Она написана на C/C++, ее исходный код открыт. библиотека включает более 1000 функций и алгоритмов. Она разрабатывается с 1998 г., сначала в компании Интел, теперь в Itseez при активном участии сообщества. О высокой популярности библиотеки свидетельствует количество загрузок, их более 6000000 загрузок (без учета svn/git трафика).

Существуют библиотеки, более продвинутые по функциональности, например, Halcon. Есть библиотеки более специализированные, делающие акцент на какой-либо конкретной задаче, например, libmv. OpenCV – самая большая библиотека по широте тематики.

Библиотека распространяется по лицензии BSD, что означает, что ее можно свободно и бесплатно использовать как в открытых проектах с открытым кодом, так и в закрытых, коммерческих проектах. Библиотеку не обязательно копировать целиком в свой проект, можно использовать куски кода. Единственное требование лицензии – наличие в сопровождающих материалах копии лицензии OpenCV.

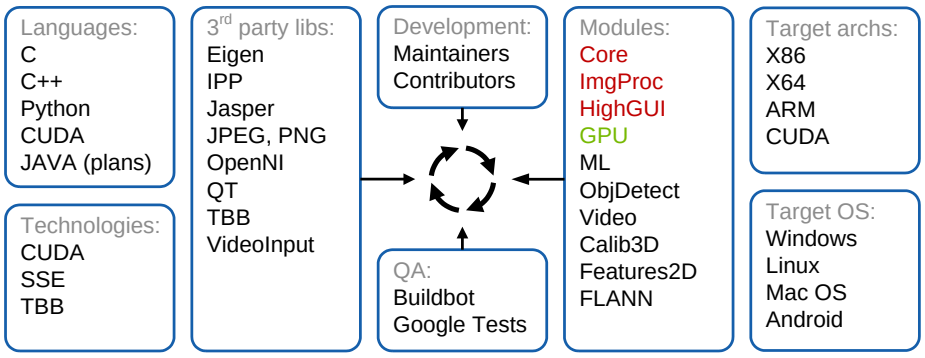
Из-за либеральной лицензии библиотека используется многими компаниями, организациями, университетами, например, NVidia, Willow Garage, Intel, Google, Stanford University. Компании NVidia и WillowGarage частично спонсируют ее разработку.

Есть основание думать, что не все компании, использующие OpenCV, афишируют это. Приведем небольшой перечень известных авторам проектов, в которых используется OpenCV:

- Система зрения робота PR2, разработанного компанией WillowGarage (www.willowgarage.com). Проект PR2 нацелен на решение сложной задачи – создание платформы для персональной робототехники.
- Аудио-визуальная инсталляция в Музее Современного Искусства (Сан-Франциско).
- Контроль качества монет, изготавливаемых Центробанком Китая.
- Курсы компьютерного зрения в Стэнфорде.
- Панорамы улиц в картах Google.

Многомерная архитектура проекта представлена на рис. Библиотека состоит из 16 модулей. Реализованы около 1000 алгоритмов. Функциональность доступна на разных языках: C, C++, Python, CUDA, Java. Поддерживаются основные операционные системы: MS Windows, Linux, Mac, Android, iOS. Есть возможность использования сторонних библиотек, например, для работы с устройством Kinect (OpenNI), разработки параллельных программ (TBB) и др.

Проект использует систему непрерывной интеграции. Каждую ночь последние рабочие версии OpenCV скачиваются из репозитория и компилируются на 50 различных конфигурациях (разные операционные системы, разные платформы, разные параметры и т. д.), после чего запускаются регрессионные тесты. Например, проводится реальное тестирование на Android-планшетах.



Основные модули библиотеки можно отнести к 4 группам (разделам):

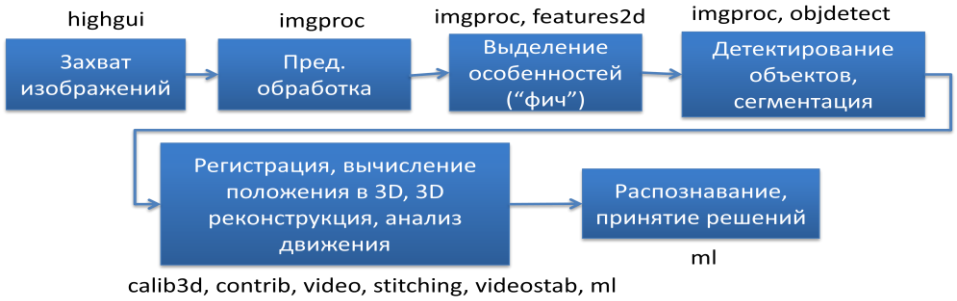
- Модули core, highgui, реализующие базовую функциональность (базовые структуры, математические функции, генераторы случайных чисел, линейная алгебра, быстрое преобразование Фурье, ввод/вывод изображений и видео, ввод/вывод в форматах XML, YAML и др.).
- Модули imgproc, features2d для обработки изображений (фильтрация, геометрические преобразования, преобразование цветовых пространств, сегментация, обнаружение особых точек и ребер, контурный анализ и др).
- Модули video, objdetect, calib3d (калибровка камеры, анализ движения и отслеживание объектов, вычисление положения в пространстве, построение карты глубины, детектирование объектов, оптический поток).
- Модуль ml, реализующий алгоритмы машинного обучения (метод ближайших соседей, наивный байесовский классификатор, деревья

решений, бустинг, градиентный бустинг деревьев решений, случайный лес, машина опорных векторов, нейронные сети и др.).

Обратим внимание на поддержку работы с xml-файлами. Результаты промежуточных вычислений можно сохранять в xml-файлах, а затем их прочитать, например, в другой программе, что облегчает разработку алгоритма по частям, работу над алгоритмом целой командой.

Заметим, что библиотека OpenCV реализует, как правило, только базовые операции, используемые в компьютерном зрении. Таким образом, ее можно рассматривать как в целом *низкоуровневую* библиотеку компьютерного зрения. Для решения серьезных задач необходимо на основе предоставленных библиотекой кирпичиков создавать свои сложные приложения.

На рисунке ниже представлена общая схема типичного приложения, предназначенного для решения той или иной задачи компьютерного зрения (конечно, не все приложения подпадают под эту схему).



Все начинается с захвата изображений (модуль highgui). Вы читаете изображение из файла или читаете видео с сетевой камеры через сетевой протокол.

Далее осуществляется предварительная обработка (модуль imgproc), такая, как устранение шума, выравнивание яркости, контраста, выделение и удаление бликов, теней. Например, один и тот же объект при разном освещении выглядит по-разному. В ярком свете красная машина, движение которой, необходимо отслеживать, будет ярко-оранжевой. В пасмурную погоду та же машина будет выглядеть красно-розовой. В этом случае на изображении необходимо выполнить выравнивание цветов. Предобработка может быть простой, но может заключать в себе целую сложную технологию.

Следующий этап – выделение особенностей (модули imgproc, features2d). Например, в задаче слежения за объектом это может быть поиск специальных точек на объекте, за которыми легко наблюдать; для задачи детектирования (т. е. обнаружения на изображении) лица – вычисления описания каждого пиксела.

Далее происходит детектирование интересующих нас объектов, выделение значимых частей, сегментация изображения (модули imgproc, objdetect). Если, например, камера неподвижна, а изображение подвижное, можно использовать алгоритмы вычитания фона.

После этого вы решаете основную задачу, такую, как вычисление расположения объекта в 3d, реконструкцию 3d структуры, анализ структуры, регистрацию и т. п. (модули calib3d, contrib, video, stitching, videostab, ml). Например в задаче склейки панорам изображений – это сопоставление частей разных кадров, определение нужного преобразования. В задаче видеонаблюдения это восстановление траекторий объектов и т. д.

В конце происходит распознавание и принятие конкретных решений (модуль ml). Например, в системе видеонаблюдения: появился нежелательный объект в кадре или нет. В задаче детектирования текста – детектирован текст, что именно за текст и т. д.

Вот вкратце базовое описание основных компонентов OpenCV и схемы работы с библиотекой. Теперь рассмотрим библиотеку более подробно.

1.2. Где взять OpenCV и достать информацию по библиотеке?

Сайт библиотеки – http://opencv.org. На нем можно загрузить библиотеку для разных платформ, найти документацию по библиотеке, краткие инструкции по установке (Quick Start), уроки по ее использованию (Tutorials), справочную карту (Cheat Sheet), которую полезно сразу распечатать, ссылки на полезные сайты и т. п. Ссылка WIKI ведет на сайт для разработчиков http://code.opencv.org.

На сайте http://docs.opencv.org собрана вся документация. В частности, имеются подробные описания всех функций с полнотекстовым поиском.

На сайте http://answers.opencv.org можно задать свой вопрос, касающийся библиотеки, и получить на него ответ от разработчиков или других пользователей. Прежде чем сформулировать свой вопрос, имеет смысл попытаться найти похожий, среди тех, которые уже задавали. Наиболее популярные вопросы попадают на верх списка. На них нужно обратить внимание в первую очередь.

1.3. OpenCV на Python’е

Для первых экспериментов с библиотекой авторы рекомендуют использовать Python (http://python.org). Программу на этом языке можно писать в любом текстовом редакторе. Далее без предварительной компиляции она выполняется интерпретатором Python. Python с

библиотекой NumPy превращается в бесплатный аналог системы MATLAB, при этом сам язык более продвинутый, чем язык MATLAB.

Мы предлагаем следующую последовательность шагов для быстрого сайта (на примере Windows):

1. Загружаем самораспаковывающийся архив с SourceForge: http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.4.3/ и распаковываем его в какой-нибудь домашний каталог.
2. Загружаем и устанавливаем Python 2.7.x с http://python.org, загружаем и устанавливаем NumPy: http://numpy.scipy.org (выбираем версию, соответствующую версии Python).
3. Копируем cv2.pyd из распакованного каталога OpenCV в <каталог Python>\Lib\site-packages.

   Можно начинать эксперименты!
4. Опционально устанавливаем редактор с поддержкой Python, например, Sublime Text 2.

1.3.1. Первая программа

Набираем следующую программу в текстовом редакторе (first.py):

```
import sys, cv2 as cv   # Импортируем системный модуль
      # Импортируем модуль OpenCV (cv2) под именем cv
img = cv.imread(sys.argv[1], 1)     # Загружаем изображение
cv.imshow("original", img)    # Отрисовываем изображение
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
      # Конвертируем цветное изображение в монохромное
gray = cv.GaussianBlur(gray, (7, 7), 1.5)
      # Добавляем размытие
edges = cv.Canny(gray, 0, 50) # Запускаем детектор ребер
cv.imshow("edges", edges)     # Отображаем результат
cv.waitKey()      # Ожидаем нажатия любой клавиши для
завершения работы
```

Программа загружает изображение, имя которого указывается в параметрах запуска программы, применяет к нему серию преобразований и отображает результат.

Копируем файл lena.jpg из <opencv>/samples/c в каталог, где находится first.py, и запускаем программу (из Far, cmd и т. п.) командой:

```
python first.py lena.jpg
```

Исходное изображение и результат показаны на рисунке.



Обратим внимание на вызов функции `cv.GaussianBlur` в коде программы, которая осуществляет размытие. Если бы его не было, на выходе получили бы слишком много ребер.

Функция `cv.waitKey()` не только ждет нажатие произвольной клавиши на клавиатуре, но и выполняет обработку сообщений от операционной системы, в частности запросы на отрисовку окна. Без этой команды окно бы не «слушалось» действий пользователя.

1.3.2. Вторая программа: добавляем видео

Настоящее компьютерное зрение начинается, когда появляется работа с «живыми» камерами. В следующей программе (second.py) добавим обработку видео. Для каждого кадра будем выполнять те же операции, что и в предыдущей программе для одного изображения.

```
import sys, cv2 as cv
cap = cv.VideoCapture(0)
      # Инициализация для захвата с веб-камеры
while True:
    ok, img = cap.read()      % Загружаем очередной кадр
    if not ok:
        break
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
      # Конвертируем цветное изображение в монохромное
    gray = cv.GaussianBlur(gray, (7, 7), 1.5)
```

```
    # Добравляем размытие
    edges = cv.Canny(gray, 1, 50)   # Детектируем ребра
    cv.imshow("edges", edges) # Отображаем результат
    if cv.waitKey(30) > 0:    # Ожидаем 30 мс
        break     # Если клавиша нажата, то выход из цикла
```

Здесь функция `cv.waitKey()` вызывается с одним входным аргументом. Это количество миллисекунд (в нашем случае 30), в течение которого система ожидает нажатия на клавишу. По-умолчанию (как в программе first.py) используется значение `0`, что означает бесконечное время ожидания. Функция возвращает код нажатой клавиши. Если клавиша не нажата, то возвращается `-1`.

Обратите внимание на небольшой размер программы, которая, тем не менее, дает довольно эффектный результат.

1.3.3. Третья программа: добавляем детектирование лиц

Добавим процедуру детектирования лиц. Получим следующий код (face.py):

```
import sys, cv2 as cv
cap = cv.VideoCapture(0)
cascade = cv.CascadeClassifier(
      "lbpcascade_frontalface.xml")
            # Загрузка обученного каскадного классификатора
while True:
    ok, img = cap.read()
    if not ok:
        break
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    sf = min(640./img.shape[1], 480./img.shape[0])
    gray = cv.resize(gray, (0,0), None, sf, sf)
            # Масштабирование
    rects = cascade.detectMultiScale(gray, scaleFactor=1.3,
        minNeighbors=4, minSize=(40, 40),
        flags=cv.cv.CV_HAAR_SCALE_IMAGE)  % Детектирование
    gray = cv.GaussianBlur(gray, (3, 3), 1.1)
            # Размываем
```

```
    edges = cv.Canny(gray, 5, 50)   # Детектируем ребра


    out = cv.cvtColor(edges, cv.COLOR_GRAY2BGR)
    for x, y, w, h in rects:
        cv.rectangle(out, (x, y), (x+w, y+h), (0,0,255), 2)
            # Вокруг найденного лица
            # рисуем красный прямоугольник
    cv.imshow("edges+face", out)
    if cv.waitKey(30) > 0:
        break
```

Сначала с помощью функции `cv.CascadeClassifier()` загружаем обученную модель (каскадный классификатор) для детектирования лиц.

Если применять алгоритм детектирования к ребрам, то лиц, скорее всего, не найдем, поэтому алгоритм применяем к исходному изображению после его преобразования в монохромное изображение и масштабирования. Использование большого разрешения в задаче детектирования лиц особого смысла не имеет, поэтому изображение масштабируем (функцией `cv.resize()`). В результате с помощью изотропного (одинакового по обоим осям) масштабирования помещаем изображение в прямоугольник размера 630×480.

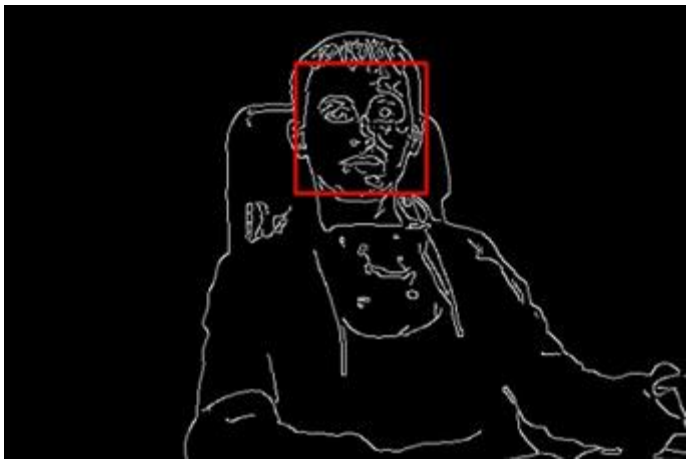
Алгоритм детектирования (детектор) вызываем функцией `cascade.detectMultiScale()`. Выделяем ребра и совмещаем изображения. Вокруг найденного лица рисуем красный прямоугольник.

Обратим внимание, что нигде не требуется явно выделять и освобождать память для хранения изображений: все происходит автоматически. Это сокращает время разработки программ и создания прототипов

Перед запуском программы копируем `lpbcascade_frontalface.xml` из `<OpenCV>/data/lbpcascades` в тот же каталог, что и `face.py`. Запускаем:

```
python face.py
```

Результат показан на рисунке.



1.3.4. Несколько замечаний об использовании связки Python + OpenCV

Использование OpenCV в связке с Python удобно для создания простых приложений, проведения первых экспериментов, создания прототипов. Заметим, что на языке Python доступна практически вся функциональность библиотеки. Слабым местом могла бы быть производительность, но если использовать «векторно-матричную» идеологию (как в MATLAB'е), то можно получать и достаточно быстрый код. Для этого следует избегать обработки изображений попиксельно и других массивов поэлементно в циклах. Вместо этого использовать векторные функции, которые работают сразу со всем изображением или массивом.

Другое хорошее свойство Python – его самодокументированность. В интерактивной сессии Python (в командном окне) можно быстро получить список доступных функций, список методов конкретного класса, быструю справку по каждой функции и т. д.:

```
>>> import cv2 as cv
>>> dir(cv) # Получаем список функций и классов в cv
>>> cv.resize.__doc__   # Описание функции resize
>>> cap = cv.VideoCapture(0)
>>> dir(cap)            # Список методов VideoCapture
>>> cap.read.__doc__    # Описание метода read
```

Обертки для Python генерируются автоматически на этапе сборки библиотеки.

Настоятельно рекомендуем ознакомиться с набором примеров на Python, которые можно найти в папке `<OpenCV>/samples/python2`: устранение размытия в результате движения камеры, распознавание рукописных цифр, видеоэффекты и т. д.

1.4. OpenCV на C/C++

Для того, чтобы использовать OpenCV на C/C++ понадобится Visual Studio (для Windows), XCode (для OS X и iOS) или компилятор GCC вместе с утилитой make (для Linnux). По возможности, не используйте MinGW, Borland C++, Sun Studio и разные экзотические компиляторы. OpenCV довольно активно использует разные возможности компиляторов и не все они адекватно справляются с этой задачей.

Также необходима утилита CMake (под Windows не обязательна, но желательна). CMake – это система сборки программного обеспечения из исходного кода. Она не выполняет саму сборку, а создает файлы (проектные или make-файлы), управляющие этой сборкой.

1.4.1. Первая OpenCV-программа на C++

Первая программа на C++ будет делать то же, что и программа first.py на Python'е.

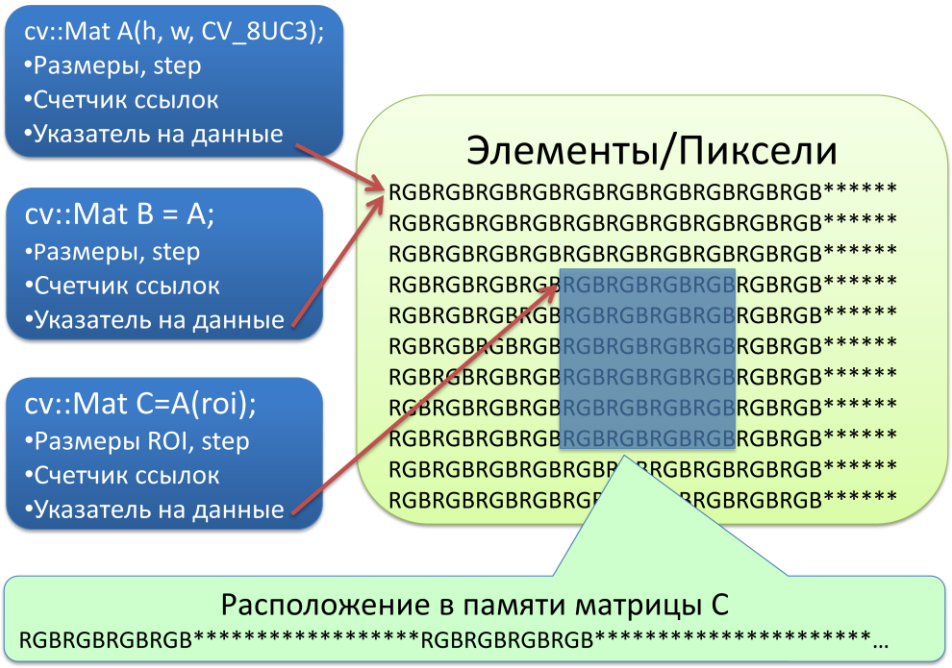
```
#include "opencv2/opencv.hpp"
using namespace cv;
int main(int argc, char** argv)
{
    Mat img, gray, edges;     // Объявление матриц
    img = imread(argv[1], 1); // Читаем изображение
    imshow("original", img); // Отрисовываем изображение
    cvtColor(img, gray, COLOR_BGR2GRAY);
      // Конвертируем в монохромный формат
    GaussianBlur(gray, gray, Size(7, 7), 1.5);
      // Устраняем размытие
    Canny(gray, edges, 0, 50);
      // Запускаем детектор ребер
    imshow("edges", edges);
      // Отрисовываем изображение
    waitKey();
```

```
        //Ожидаем нажатия клавиши
    return 0;
}
```

Как скомпилировать и запустить эту программу см. инструкции в учебнике http://docs.opencv.org/doc/tutorials/introduction/table_of_content_introduction/table_of_content_introduction.html.

1.4.2. Работа с матрицами

Как было сказано ранее, если хочется провести небольшой эксперимент с OpenCV, как правило Питон будет наилучшим выбором. К C++ же имеет смысл обращаться, когда хочется сделать свой алгоритм работающий непосредственно с пикселями изображений, и делающий это эффективно, или для реализации уже конечного приложения, не прототипа. В любом случае, как только вы начинаете использовать OpenCV из C++, вы немедленно столкнетесь с типом `cv::Mat` (матрица), который в OpenCV используется для хранения почти всех объектов – изображений, массивов точек, векторных полей вычисляемых алгоритмами оптического поток, обычных матриц и т.д. Собственно, это аналогично MATLAB, где почти все данные хранятся в массивах. Итак, класс `cv::Mat` реализует многомерные многоканальные массивы.



При его использовании вы освобождаете себя от забот по управлению памятью: выделению и освобождению. Класс реализует счетчик ссылок и память освобождается только тогда, когда она больше не нужна.

Изображение – это двумерный массив. Все его элементы расположены подряд по строкам. Кроме числа строк и числа столбцов имеется параметр `step`, равный расстоянию между строками. Этот параметр не меньше числа столбцов, но может быть его больше. В последнем случае в конце каждой строки имеется неиспользуемое пространство.

Предположим, вы создали некоторую матрицу (матрица A на рисунке выше), указав число ее строк и столбцов и опционально начальное значение для элементов матрицы. Параметр `step` вычисляется автоматически, выделяется буфер память нужного размера, при необходимости заполняется заданным значением. Счетчик ссылок приравнивается к 1. Если эта матрица присваивается другой матрице (матрица B на рисунке), то физического копирования данных не происходит, но на 1 увеличивается счетчик ссылок, а указатель данных указывает на то же место в памяти. Как только вы завершаете работу с каким либо объектом, счетчик ссылок уменьшается на 1. При достижении значения 0 память освобождается.

Допустим, вы нашли лицо на изображении и хотите найти на нем глаза. В этом случае вам нужно не все изображение, а только его часть, соответствующая лицу. Для этого удобно создать матрицу, являющуюся частью другой матрицы, при этом можно избежать копирования данных (матрица C на рисунке). Заметим, что параметр step для такой подматрицы будет равен параметру step исходной матрицы. Указатель на данные будет указывать на начало подматрицы. Счетчик ссылок после создания такой подматрицы также увеличивается на 1.

Если необходимо настоящее копирование данных, нужно использовать метод `clone()`, например:

```
cv::Mat M2 = M.clone();
```

Любому элементу матрицы можно присвоить значение. Делается это с помощью метода `.at(y,x)`, где `y` – номер строки, `x` – номер столбца. Рассмотрим небольшой пример.

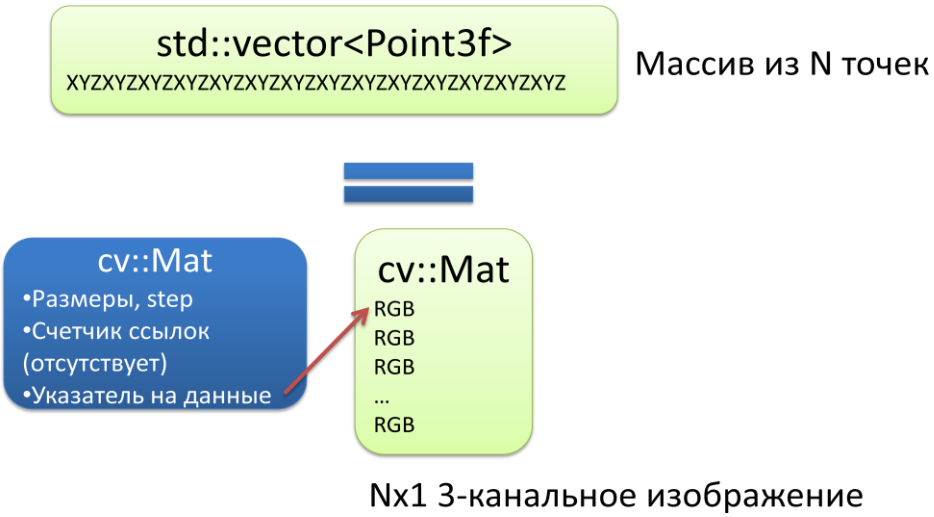
```
Mat M(480,640,CV_8UC1); // Создаем изображение 640x480
Rect roi(100, 200, 20, 20);   // Определяем ROI
Mat subM = M(roi);      // Выделяем ROI в отдельную матрицу
subM.at<uchar>(y,x)=255;
      // Изменяем пиксель (x,y) в subM и (x+100, y+200) в M
```

Так как в данном примере данные одни и те же, то изменяя подматрицу `subM`, вы изменяете и матрицу `M`.

Продолжим этот пример и покажем, как работать с итераторами. Пусть в выбранном ROI необходимо оценить «резкость», определяемую как сумму модулей градиентов, вычисленных в каждом пикселе. Эта задача может быть полезной для реализации автофокуса (автофокус соответствует локальному максимуму функции «резкости»). Необходимо пробежаться по всем пикселам, найти в каждом из них градиент, вычислить его модуль и прибавить к сумматору. Для того чтобы пройтись по всем элементам матрицы, удобно использовать итераторы, похожие на итераторы из стандартной библиотеке stl:

```
Mat_<Vec3b>::iterator it= subM.begin<Vec3b>(),
                      itEnd = subM.end<Vec3b>();
float contrast = 0.f;
for(; it != itEnd; ++it)
{
   uchar* ptr = &(*it);
   int dx = ptr[1] – ptr[-1],
       dy = ptr[subM.step] – ptr[-subM.step];
   contrast += sqrtf((float)(dx*dx + dy*dy));
}
```

Кроме матриц библиотека OpenCV умеет работать с обычными STL векторами. При этом внутри библиотеки вектора часто обрабатываются как обычные матрицы (cv::Mat). То есть, для входного вектора внутри библиотеки дополнительно создается заголовок cv::Mat, который указывает на те же данные (данные не копируются). По завершении обработки временный заголовок удаляется.



1.4.3. Пример: поиск плоских объектов

В качестве практического примера использования OpenCV с C++ рассмотрим задачу поиска на изображении плоских объектов. Пусть имеется некоторое изображение плоского объекта, например, передняя грань коробки. Требуется найти этот объект на другом большом изображении некоторой сложной сцены. Коробка может быть загорожена другими объектами, ориентирована произвольным образом в пространстве. Освещение сцены может быть иным.

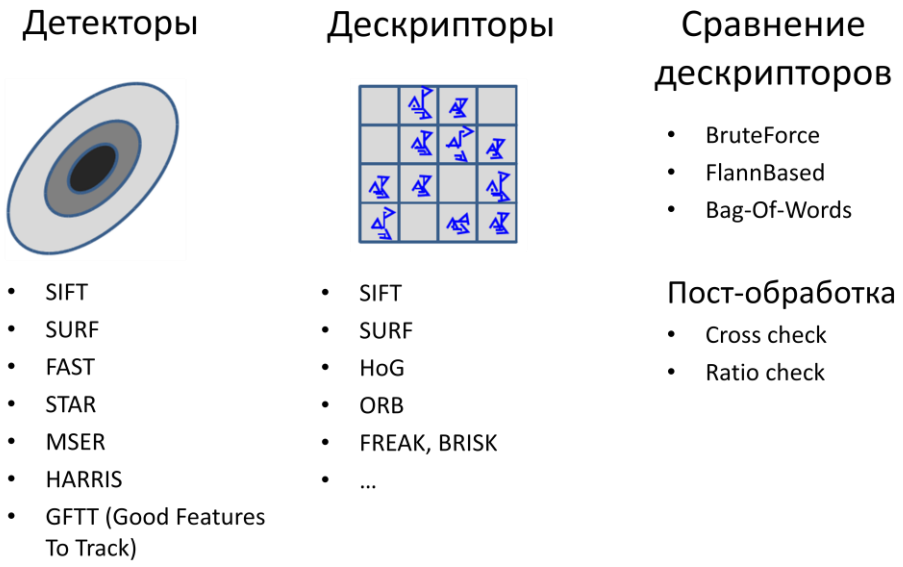




Основная идея метода – нахождение особых точек, т. е. точек, примечательных для данного объекта, за которые можно «зацепиться». Однородные участки поверхности для идентификации объекта обычно не подходят, так как могут совпасть с чем угодно. Если возьмем окрестность

более интересных точек – то случайное совпадение маловероятно. Мы находим точки, строим их описание (дескрипторы), и пытаемся найти похожие точки на втором изображение. После чего отфильтровываем неправильно сопоставленные точки, исходя из гипотезы что объект жесткий (недеформируемый) и поэтому все точки на нем должны преобразовываться от одного кадра к другому согласованно. В случае плоского объекта это должно быть перспективное преобразование (гомография).

Все необходимое для реализации этого метода есть в модуле features2d. Он позволяет находить особые точки, строить их дескрипторы, применять методы сравнения дескрипторов.



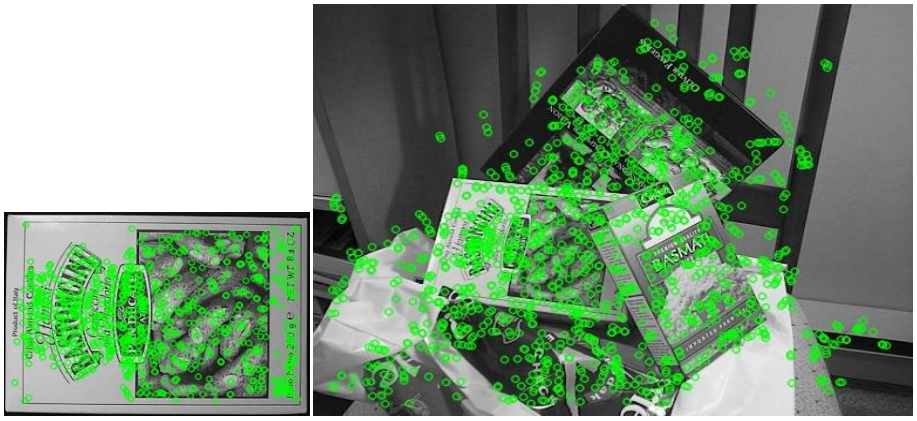
Приведем код, решающий данную задачу:

```
// Загружаем изображения:
Mat img1 = imread(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
Mat img2 = imread(argv[2], CV_LOAD_IMAGE_GRAYSCALE);
// Находим особые точки на каждом изображении
// Вычисляем их дескрипторы:
Ptr<Feature2D>
    surf=Algorithm::create<Feature2D>("Feature2D.SURF");
vector<KeyPoint> keypoints1, keypoints2;
Mat descriptors1, descriptors2;
surf->operator()(img1, Mat(), keypoints1, descriptors1);
```

```
surf->operator()(img2, Mat(), keypoints2, descriptors2);
// Находим наилучшее соответствие между особыми точками
// на изображениях
vector<DMatch> matches;
BFMatcher(NORM_L2, true).match(descriptors1, descriptors2,
    matches);
// Находим оптимальное преобразование,
// согласующееся с большинством пар точек.
vector<Point2f> pt1, pt2;
for( size_t i = 0; i < matches.size(); i++ ) {
     pt1.push_back(keypoints1[matches[i].queryIdx].pt);
     pt2.push_back(keypoints2[matches[i].trainIdx].pt);
}
// H – это матрица оптимального перспективного
// преобразования от img1 к img2
Mat H = findHomography(pt1, pt2, RANSAC, 10);
```

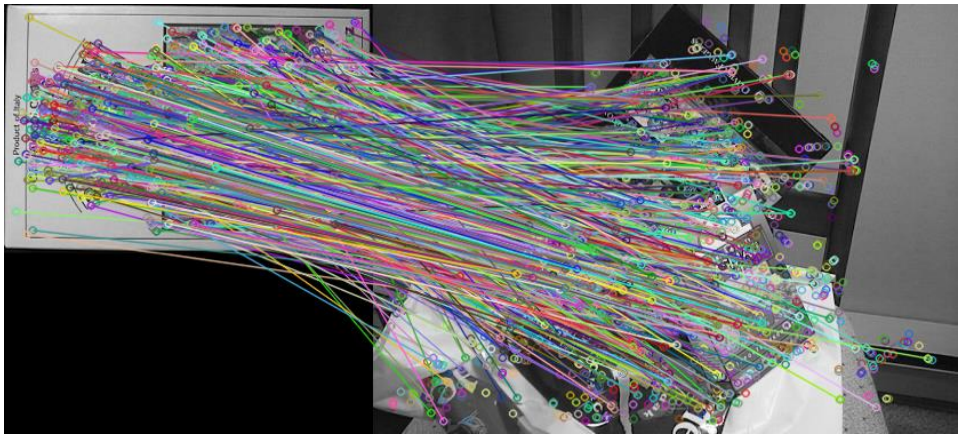
Мы загружаем оба изображения. Находим особые точки и вычисляем их дескрипторы. При этом создаем экземпляр алгоритма `Feature2D`, используя параметры по умолчанию. В результате получаем 2 «списка» особых точек и 2 «списка» их дескрипторов: `keypoints1`, `keypoints2`, `descriptors2`, `descriptors2`, при этом `keypoints1`, `keypoints2` – это векторы, содержащие номера особых точек, а `descriptors2`, `descriptors2` – матрицы, строки которых содержат их дескрипторы.

На рисунке указаны найденные особые точки.



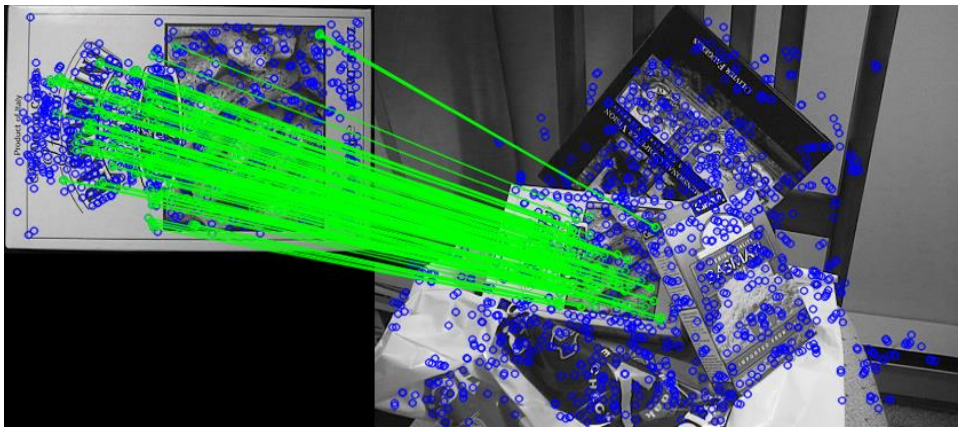
После этого, сравнивая каждый набор дескрипторов из первого изображения, с каждым таким набором из второго изображения, находим наилучшее соответствие между особыми точками.

На рисунке графически изображено найденное соответствие.



Далее находим матрицу наилучшего перспективного преобразования (гомографии), осуществляющую это соответствие.

На рисунке отмечены «правильные» пары особых точек, т. е. пары, подчиняющиеся найденному перспективному преобразованию. Обратите внимание на пару точек (самый верхний отрезок), найденную ошибочно (эти точки принадлежат разным объектам), но геометрически не противоречащую остальным парам.



После того как матрица гомографии найдена само расположение коробки в пространтсве можно найти с помощью функции `solvePnP()`.

Заметим, что данный метод работает также для склейки панорам, если снимаемая сцена (например пейзаж) находится достаточно далеко от точки съемки, чтобы считать ее плоской.

В случае неплоских объектов, «правильные» пары можно найти с помощью `findFundamentalMat()` вместо `findHomography()`, после чего опять использовать `solvePnP()`.

2. Литература

1. Сайт OpenCV: http://opencv.org, где можно найти ссылки на документацию, уроки, исходный код, сайт вопросов, wiki и т.д.
2. Лучшая современная книга по компьютерному зрению (на английском) : http://szeliski.org/Book/
3. «Официальная» книга про OpenCV: http://www.amazon.com/Learning-OpenCV-Computer-Vision-Library/dp/0596516134 (также доступна на Озоне. описывает OpenCV 1.x!). Сейчас завершается работа над вторым изданием
4. Примеры кода из книги выше: http://examples.oreilly.com/9780596516130/
5. «Официальный» учебник по Питону на русском: http://ru.wikibooks.org/wiki/Учебник_Python_2.6
6. Numpy – превращает Питон в аналог Матлаба (и необходим для использования OpenCV с Питоном): http://docs.scipy.org/doc/