Введение в OpenCV

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

17 сентября 2018 г.

Версия: 0.12

Аннотация

В данной заметке начнется твое знакомство с библиотекой opencv.

Цель: Формирование растровых изображений. Рисование кривых, в частности, текста на изображениях (картинках). Изменение яркости, контраста и цветового пространства (представления) изображений.

1 Введение

В данной заметке будет показано как формировать изображения, путем рисования, в частности, изменять существующие изображения. Также будет проиллюстрировано как вводить/считывать изображения, накладывать изображения руг на друга. Выполнять отрисовку линий, окружностей и текста. Изменять яркость, контраст и цветовое пространство.

1.1 Первая программа

Далее показано как создать изображение с фразой приветсвующей мир.

Создание и показ изображения Начнем с первой программы, которая создает изображение размера 500 на 400, т.е. 500 пикселов в ширину и 400 в высоту и, как аналог при изучении языков програмирования, выводит на нем соответсующую традиционную фразу – приветствие в миру:

```
1 #include <opencv2/core/core.hpp>
2 #include <opencv2/highgui/highgui.hpp>
```

```
3  #include <string>
4
5  using namespace cv;
6  using namespace std;
7
8  int main()
9  {
10   Mat img(400, 500, CV_8UC3);//Создание изображения
11   //имеющее размер 500 в ширину на 400 в высоту.
12
13   //Для удобства, аргументы вызываемой функции:
14   string text = "Hello World!";//Целевая фраза.
15   Point textOrg(100, img.rows/2);//Местоположение.
16   int fontFace = FONT_HERSHEY_SCRIPT_SIMPLEX;//Фонт.
17   double fontScale = 2;//Его масштаб, размер.
18   Scalar color(200,100,50);//Цвет текста.
19
20   //Далее выполняется отрисовка фразы из переменной
21   //text в изображение img в положение textOrg.
22   //Фонт/гарнитура задается следующими переменными:
23   //fontFace, fontScale, thickness.
24   putText(img, text, textOrg, fontFace, fontScale,
25     color);
26
27   imshow( "My World", img );
28   waitKey(0);
29   return 0;
30 }
```

Рассмотрим далее более детально данный код.

Заголовочные файлы Строчка 1 ранее приведенной программы ссылается на заголовочный файл `core.hpp`, отвечающий за базовый набор функций предоставляемый библиотекой OpenCV, а строчка 2 – на `highgui.hpp`, содержащий функции для взаимодействия с графическим движком операционной системы: считывание и запись графических файлов, а также взаимодействие с графическим оконным интерфейсом.

Функциональность программ дополняется соторрними функциями из зругих библиотек. Библиотеки определяются своими заголовочными файлами. К наиболее важным в опенсв следует отнести следующие:

- `opencv2/core/core.hpp` – определяет как базовые объекты (например, классы) библиотеки OpenCV: изображение, цвет и др, так и функции: преобразование самих изображений и их цветов;
- `opencv2/highgui/highgui.hpp` – определяет кросс-платформенные функции взаимодействия с оконной системой;
- `opencv2/imgproc/imgproc.hpp` – определяет основные/традиционные функции цифровой обработки изображений: отрисовка кривых и тому подобное.

Весь код, в частности, упомянутый последним предполагает кросс-платформенность, т.е. возможность сборки под разными операционными системами и, соответственно, идентичную работу программы. Другим подходом являлся бы использования функций конкретной операционной системы: X-Windows для *nix систем или WinAPI для Windows. Такой подход дал бы больший контроль над выводом графической информации в ущерб возможности быстрого переноса код между системами.

В рамках данных заметок будем придерживаться крос-платформенного подхода.

Растровое изображение В стр. 10 показано как создавать растровые изображение. В рамках библиотеки OpenCV считается, что изображение – это матрица, элементы которой соответствуют пикселям. Именно по этой причине размер изображения задается как это традиционно принято для матриц: высота на ширину (а не как это принято для изображений: ширина на высоту). Последний аргумент конструктора указывает на тип данных. В данном случае `CV_8UC3` указывает на 8 битовое без-знаковое целое число на компоненту коих 3, т.е. задано традиционное 24-битое цветное изображение.

Более подробно данный объект будет описан в подразделе 1.2.

Отрисовка текста В строчках 14 - 18 производится инициализация переменных отвечающих за аргументы функции putText, которая выводит текстовую строчку в заданное место изображения. Это сделано для удобства чтения программы. Так, переменная `text` (см. стр. 14) содержит текстовую строчку, которую необходимо нарисовать на изображении в точке заданной переменной `textOrg` (см. стр. 15). Для отрисовки текста используется гарнитура текста заданная переменной `fontFace` (см. 16), где размер задается переменной `fontScale` (см. стр. 17), а цвет – `color` (см. стр. 18).

Все эти переменные передаются в качестве аргументов функции putText в строчке 24, первым аргументом которой как раз указывается изображение (в данном случае, переменная `img`) на котором будет произведена отрисовка текста согласно ранее указанным параметрам.

Другие функции отрисовки описаны в подразделе 1.7.

Интерактивное взаимодействие Выполняемая программа должна как-то взаимодействовать с пользователем (иначе она бессмысленна). Как далее будет показано, можно, например, изображения сохранять/считывать в/из файл(ы)/ов. Но это долго (сохранил, потом посмотрел через внешнюю программу) и не всегда удобно. Поэтому уже в данной первой программе показано как выводить изображения на экран в окошко и ожидать нажатие клавиши от пользователя.

Изображение отрисовывается в окне строчкой 27, где первым аргументом указывается его название. Второй аргумент задает требуемое для отображения в создаваемом окне изображение. В данном случае размер окна будет подстроен под размер изображения, т.е. если оно будет громадным, то оно вылазит за пределы твоего экрана.

Более побробно описание взаимодействия с оконной системой будет описано в подразеделе 1.3.

Если написать программу без вызова функции `waitKey`(см. стр. 28), то программа завершит свою работу сразу после отображения изображения. Последнее означает, что окошко с изображением будет кратковременно показано, а потом сразу все закроется. Дабы это предотвратить как раз и вызывается функция `waitKey`, которая ожидает ввода любой клавиши. Вместо неё конечно можно было бы просто вызвать `scanf`, но так с точки зрения библиотеки OpenCV более корректно и правильней. Подробности также даны в подразеделе 1.3.

Упражнения Далее следует список упражнений дабы можно было поиграться с данной программой.

Упражнение. Поиграйся с программой изменяя значения переменных отвечающих за аргументы функции `putText`.

Упражнение. В цикле обновляй изображение, например, напечатав обратный отсчет по нажатию клавиши.

1.2 Объект матрица Mat

Базовым объект библиотеки OpenCV является матрица, который соответствует сущности растрового изображения. Без него фактически никак.

Растровое изображение (в отличие от векторных) состоит из пикселей. Пиксель отвечает за цвет изображения в определенной его точке и задается фиксированным набором чисел (числовых характеристик 1 или 3 или 4). Изображение сводится к матрице элементы которой соответствуют пикселям, т.е. они могут быть векторами, например, размера 3. Ввиду того, что изображений это матрица, над изображениями доступны все стандартные операции относящиеся к матрицам:

сложение, вычитание, увеличение на константу и тому подобное. Данная мысль будет развита в разделе 1.5.

Матрицу можно создавать безотносительно изображения, тогда её элементы задают уже не обязательно цвет. Поэтому, в качестве её элементов можно использовать вектора произвольного размера, но определенного типа.

Тип данных Данный параметр влияет с одной стороны на точность, а с другой на скорость обработки. В рассматриваемой программе достаточно целых 8-битовых чисел. В случае каких-то серьезных вычислений (например, Фурье преобразование) возможно потребуется использовать другие типы данных, например, числа с плавающей точкой.

Пиксель, а точнее элемент матрицы, является вектором определенного типа имеющий фиксированный размер.Размер вектора определяется типом изображения, а точнее выбранным цветовым пространством для данного изображения, типом пикселей. Так, для цветных изображений используются вектора длины 3 (например, соответствующие цветовым компонентам красный, зеленый и синий), а для монохромных (например, градации серого цвета) – вектора длины 1 (например, соответствующие просто яркости). Для собственых типов размер вектора может быть произвольным.

Для наиболее часто используемых типов в библиотеки OpenCV заданы константы. Приведем самые важные для данного курса:

- `CV_8UC3` – три 8 битных без-знаковых числа (вектор размера 3). Традиционно ещё называется 24 бита на пиксель.
- `CV_8C3` – ...

Вообще таких констант много. Более того, их можно сформировать самостоятельно посредством следующей конструкции:.

Создание Объект – матрица – создается посредством конструктора класса `Mat` (см. строчку 10 в первой программе). Первые два аргумента указывают её размер: количество строчек и столбцов. Третий аргумент указывает на тип элементов матрицы.

Выбрав определенный тип данных изображение можно создавать так:

```
1 ...
2 Mat img(400, 500, CV_8C1);//Создание изображения
3 //имеющее размер 500 в ширину на 400 в высоту.
4 //Тип пикселей: 8-битные знаковые числа.
5 ...
```

Можно создаваемый объект сразу и инициализировать, т.е. заполнить определенным значением.

::zeros ::ones ::eye как в matlab, octave

Инициализация через vector.

Копирование Как можно меньше происходит копирование. Его фактически нет. Идет счетчик ссылок.

Класс сам забоится о выделении и освобождении памяти. Состоит из header и data

Объект матрица (Mat) состоит из двух частей: массив данных, содержащий значение самих элементов матрицы, и заголовка – размер и *регион интереса*. С точки зрения C++ или иначе говоря объектно ориентированного подхода при создании новой матрицы (`Mat`) на основе текущей данные (т.е. массив элементов) всегда остаются теме же, а вот заголовок меняется. Это означает, что все изменения с элементами матрицы у одного объекта Mat будет проявляться и у другого. Так, если перед пустой строчкой 12 вставить строчку

```
1  Mat img2(img);//Создание изображения по img.
```

Далее, если ещё заменить в строчке 27 переменную img, на img2, то программа тем не менее выведет на экран тот же самый ответ (изображение с приветствием к миру). Хотя отрисовка по-прежнему делается в Img.

Для создание истинной копии нужно в явном виде вызвать метод clone.

```
1  Mat img2=img.clone();//Создание независимого
2  //изображения по изображению img.
```

Или посредством функции `toImage`.

Но тогда приветсвия пропадет.

Регион интереса Данная возможность позволяет ограничить допустимую область изменения изображения. Фактически мы создаем для изображения новые границы в рамках старого изображения (какбы нерационально используем память, фактическая длина строк гораздо больше ширины изображения...).

```
1 Rect roi( 110, 80, 100, 100);// Задаем саму область.
2 // Создаем изображение с новыми границами.
3 Mat img2 = img( roi );
```

Тогда, в частности, все функции отрисовки будут обрезаться данной областью. Например, если при клонировании ограничит и область, то надпись будет обрезана. Боле того в зависимости от того какое изображение подается функции отри-

совки будет: либо выведена только обрезанная часть, выведено все изображение с обрезанной частью.

Упражнение. Проверить последнее утсверждение.

Надо понимать, что данной свойство будет выполнено для всех функций. Все далее приведенные преобразования будут также ограничиваться таким способом данной областью. Например, повышение яркости, контраста, добавление шума и тому подобное.

Rect Range::all()

copyto.

1.3 Оконная система

Окна В первой программе окно было создано автоматически при его отрисовки. На самом деле его можно создать в явном виде, что позволит указать и тем самым настроить один важный параметр. Так, как показано ниже, окно задается вызовом функции `namedWindow`:

```
1 namedWindow( "Display window", WINDOW_AUTOSIZE );
```

В строчке 1 показано как создавать окно. Первый аргумент вызываемой функции `namedWindows` задает название создаваемого окна, но на самом деле более правильней считать, что он же является и хэндлером, т.е. данное имя указывает на объект соответствующий окну. Второй аргумент как раз настраивает параметр создаваемого окна. Оно может принимать следующие значения:

- WINDOW_AUTOSIZE – автоматически изменяет свой размер под размер изображения;
- WINDOW_NORMAL – окно определенного размера, который можно менять мышкой. Изображение вписывается в окно, т.е. масштабируется под текущий размер.

В данном случае, оно будет изменять свои размеры под показываемое изображение. Отмечу, что данный аргумент является необязательным параметром (по умолчанию – WINDOW_AUTOSIZE).

Строчку кода 1 можно вставить в первую программу перед строчкой 27. В таком случае к первому аргументу функции `imshow` следует отнестись как к хэндлеру ранее созданного окна.

moveWindow

Упражнение. Создай больше одного окна и изображения.

Ползунок trackbar, callback

Дополнительные возможности waitKey Помимо графической взаимодействия с пользователем, необходимо уметь и понимать какую клавишу нажал пользователь. Так, например, можно обработать нажатие таких клавиш как верх, вниз, влево и вправо что может позволить например управлять перемещением курсора в окошке. Данная возможность позволит создать интерактивную программу. Например, двигать объект по картинке.

Легче всего определять коды соответствующие этим клавишам путем отладочной печати.

В скобках можно указать в миллисекундах максимальное время ожидания. При этом 0 обозначает бесконечное ожидание.

В любом случае даже при последнем значении функция завершает свою работу при нажатии любой клавиши.

1.4 Ввод/вывод изображений

Изображения можно не только формировать в программе, но и загружать из файла и записывать в файл.

Вывод Помимо вывода на экран, изображение можно вывести (т.е. записать) и в файл. Запись изображения в файл осуществляется функцией imwrite (от слова image write):

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <string>
using namespace cv;
using namespace std;
int main( int argc, char** argv )
{
        Mat img(400, 500, CV_8UC3);
        string text = "Hello World!";
        Point textOrg(100, img.rows/2);
        int fontFace = FONT_HERSHEY_SCRIPT_SIMPLEX;
        double fontScale = 2;
        Scalar color(200,100,50);

        putText(img, text, textOrg, fontFace, fontScale,
                color);

        //Запись изображения img в файл ./output.bmp:
        imwrite("./output.bmp", img);
```

```
20          return 0;
21  }
```

Замечу, что в данном случае, мы только выводим изображение в файл и нет необходимости его вывода на экран (отсутствует строка 27 предыдущей программы). Тем не менее заголовочный файл (стр. 2) необходим для вызова функции `imwrite`.

Первый аргумент функции `imwrite` указывает на имя изображения, путь к выходному файлу. Как обычно, путь может быть как относительным ("./lena.bmp"), так и полным ("/home/vasay/girls/lena.bmp").

Расширение файла (например, ".bmp") фактически указывает на формат хранения изображения. Изображения может быть записано для широкого круга форматов, но все зависит от сборки библиотеки. Обычно поддерживаются традиционные форматы: bmp, jpeg, png и другие.

Цветовое пространство файла зависит от формата матрицы. Так, если элементы матрицы являются просто числами, то итоговое изображение будет монохромным. Если же вектора длины 3, то – скорее всего трехкомпонентное цветное изображение в формате RGB (red – красный, green – зеленый, blue – синий), но может быть в пространстве HSV (hue – тон, saturation – насыщенность, value – значение). При необходимости к изображению добавляется и альфа канал (прозрачность). Можно также выполнить и более тонкую настройку кодека (3й необязательный аргумент).

Записанное программой выходное изображение ("./output.bmp") можно просмотреть стандартными программами просмотра изображений.

Упражнение. Воспользуйся переменными argc и argv для определения входного файла, который необходимо преобразовать в серое. Запиши монохромное изображение добавив некий суффикс к его имени.

Ввод Помимо вывода программа должна уметь и вводить данные, т.е. в нашем случае изображения. Ввести изображение, а точнее считать его из файла, можно посредством функции `imread`:

Первый аргумент указывает путь к изображению, а второй необязательный аргумент требуемое цветовое пространство у итогового изображения, т.е. возвращаемой матрицы:

- `IMREAD_COLOR` – цветное RGB изображение,
- `IMREAD_GRAYSCALE` - монохромное изображение.

Функцией чтения стандартно поддерживается широкий набор форматов изображения: bmp, jpeg/jpg, png.

Цветовое пространство указывается посредством предопределенной константы. В частности, константа `IMREAD_COLOR` соответствует цветному изображению, т.е. пикселям состоящим из трех компонент: красный, зеленый и синий.

```
1  #include <opencv2/core/core.hpp>
2  #include <opencv2/highgui/highgui.hpp>
3  #include <string>
4  using namespace cv;
5  using namespace std;
6  int main()
7  {
8          Mat img;//Создание объекта под изображение.
9          //Считывание изображения из файла.
10         img = imread("./lena.bmp", IMREAD_COLOR);//и
11         //инициализация им матрицы img.
12
13         imshow( "Display window", img );
14         waitKey(0);
15         return 0;
16 }
```

Другая важная константа, `IMREAD_GRAYSCALE`, соответствует цветовому пространству состоящему из градаций серого цвета, тем самым компонента будет состоять из одного значения: градации серого цвета. Это позволяет при считывании изображения сразу же его преобразовать в монохромное.

Метод `empty` может быть использован для проверки успешности считывания изображения. В случае неудачи объект `Mat` будет пустым. Или img.data/// т.е. корректный способ считывать изображение такой: ...

Упражнение. Выведи серию изображений, а именно – пусть имеется набор из изображений типа `image_00.jpeg`, ..., `image_99.jpeg`. Необходимо последовательно вывести его на экран. При необходимости серию изображений можно сформировать исходя из ранее представленого материала.

argc argv[]!!

1.5 Операции над изображениями

В данном подразделе показано как выполнять простейшие преобразования над изображениями: увеличение яркости, контраста, сложение.

Матрицы С точки зрения программирования в данной библиотеки изображения по сути являются матрицами. Над матрицами доступны соответствующие опе-

рации: сложение, умножение на константу. Последнее позволяет данные операции применить к изображениям.

С другой стороны, преобразованиям над изображениями ранее обозначенные во введении к подразделу соответствуют операции над матрицами. Покажем далее как они соотносятся.

Изменение контраста Изменение контраста изображения осуществляется путем домножения всех значений изображения на константу. С точки зрения матричных операции этому соответствует умножение искомой матрицы на некую константу.

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <string>
using namespace cv;
using namespace std;
int main()
{
        Mat img;
        img = imread("./lena.bmp", IMREAD_COLOR);
        img *= 1.1;

        imshow( "Display window", img );
        waitKey(0);
        return 0;
}
```

Упражнение. Что будет происходить с увеличением числа (1.1)? Обосновать результат. Что будет при уменьшении?

Как каждую компоненту по отдельности? Что с отрицательным числом?

Изменение яркости Увеличению яркости изображения осуществляется путем добавления некой константы (определенного цвета) ко всему изображению. С точки зрения матричных операции этому соответствует добавление к искомой матрице некого элемента.

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <string>
using namespace cv;
using namespace std;
```

```
 6 int main()
 7 {
 8         Mat img;
 9         img = imread("./lena.bmp", IMREAD_COLOR);
10         img += 30;
11         namedWindow( "Display window", WINDOW_AUTOSIZE );
12         imshow( "Display window", img );
13         waitKey(0);
14         return 0;
15 }
```

Упражнение. Что будет происходить с увеличением числа (30)? Обосновать результат. Тоже что и раньше... уменьшение и отрицательные.

Для правильного добавления константы нужно число преобразовать в правильный объект:

```
1         img += Scalar(30);
```

Упражнение. Проделать прошлое упражнение заново.

Упражнение. Добавить ползунок который интерактивно меняет яркость изображения.

Цвет !!Может ранее Scalar (значение пикселя)? 4 элементный. Point...

Воспринимаемый человеком цвет задается тремя числами, компонентами. Традиционно ими являются: красный, зеленый и синий. В библиотеки опенсв они по историческим причинам (связано с порядком байт в 4 байтовых величинах) задаются в обратом порядке: синий, зеленый и красный. Так, ранее примененная сущность `Scalar` позволяет задавать цвет тремя числами заданными именно в таком порядке.

Исходя из последнего, чтобы добавить ко всему изображению оттенок красного цвета нужно сделать так:

```
1         img += Scalar(0, 0, 30);
```

Упражнение. Добейтесь идентичного результата с программой из прошлого параграфа.

Область определения. Тип целый/вещественный. Диапазон изменения. saturate

Сложение Преобразования можно выполнять не только на единичных изображениях, но и на парах. Так, можно сложить два изображения, что позволит их наложить друг на друга. С точки зрения матричных вычислений данному будет соответствовать сложение матриц.

`addWeighted` форирует взвешенную сумму двух изображений.

min max

1.6 Преобразование цветов

Цветные пространства. Разная цель. Интенсивность, rgb? hsv...

Показана как можно преобразовывать цвета изображения. Данная операция важна при распознавании образов ввиду упрощения изображения (отбрасывания цветовой части данных). Также это является цветовым фильтром, который обладает некой эстетикой. Цвета также позволяют отделить на изображении определенный цвет, например, цвет кожи.

С точки зрения математики рассматриваемые ранее преобразования действовали над всеми компонентами всех пикселей по отдельности. Данные операции действуют "внутри" пикселя, т.е. являются операцией над компонентами (внутри пикселя).

Градации серого Ранее было отмечено, что изображение можно сразу загрузить из файла в монохромном цветовом пространстве указав соответствую константу в функции `imread`. Но можно выполнить преобразование цветного изображения в градации серого в явном виде.

```
1  #include <opencv2/core/core.hpp>
2  #include <opencv2/highgui/highgui.hpp>
3  #include <string>
4  using namespace cv;
5  using namespace std;
6  int main( int argc, char** argv )
7  {
8          Mat img;
9          img = imread("./lena.bmp", IMREAD_COLOR);
10         Mat img_gray;//Создание объекта под выходное
11         //изображение. В него будет записан результат.
12         //Преобразуем изображение из цветного в серое.
13         cvtColor( img, img_gray, COLOR_BGR2GRAY );
14         //Записываем серое изображение в файл.
15         imwrite("./lena_gray.bmp", img_gray);
16         return 0;
17 }
```

В данном примере была вызвана функция `cvtColor` первый аргумент которой указывает на исходное изображение, второй на результирующее, а третий на тип преобразования. Константа `COLOR_BGR2GRAY` отвечает за преобразование цветного трехкомпонентного изображения в монохромное.

Формула....

Негатив Для негатива нужно "обратить цвета"(т.е. соответствующие компоненты), а именно – яркая компонента должна стать тусклой, а тусклая яркой. Минимальная яркость является 0, а максимальная 255. Фактически необходимо вычесть из матрицы состоящей одних числе 255 искомую матрицу.

Как были сказано ранее ввиду того, что изображения представляются в виде матриц, над изображениями доступны операции, которые предназначены для матрицы. В частности, вычитание. Тогда программа превратиться в:

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <string>
using namespace cv;
using namespace std;
int main()
{
        Mat img;
        img = imread("./lena.bmp", IMREAD_COLOR);
        Mat img_neg;//Создание объекта под выходное
        //изображение. В него будет записан результат.
        img_neg = Scalar(255) - img;//Вычисляем негатив.
        namedWindow( "Display window", WINDOW_AUTOSIZE );
        imshow( "Display window", img );
        waitKey(0);
        return 0;
}
```

Монохромное Монохромность подразумевает, что в изображении вместо градаций серого используются градации какого-то другого цвета. Опять же мы можем воспользоваться математическими операциями для достижения нужной цели.

Сепия Сепия это коричневатый цвет. Изображение можно превратить в монохромное, но в оттенках сепии. Для нужно сначала превратить изображение в монохромное, а потом умножить все компоненты на цвет сепии. Далее нужно усреднить изображением из градаций серого.

Отображение серого на цвета Данное преобразование позволяет лучше отобразить серые (монохромные) изображения. К таким изображениям относятся исходные изображения с большой битностью и как результат неких преобразований (например, плвающая точка). Например, модуль Фурье преобразования или производная изображения. Использования обычной серой шкалы не позволит увидеть мелкие детали в изменении интенсивности.

Последнее связано с тем, что всего таких градаций обычно (на 8-битных мониторах) 255. Более того, и у человека есть и свои ограничения. Отображение монохромных изображений на цветные позволит существенно расширить данную битность. Данное преобразование фактически уплотняет цвета.

Осуществляется данное преобразование функцией `applyColorMap`:

```
1 Mat img_dst;// Не нужно инициализировать!
2 applyColorMap(img_src, img_dst, COLORMAP_HOT);
```

Первый аргумент задает исходное изображение, второй целевое изображение, при этом, оно будет сформировано, а последний аргумент задает конкретный тип используемого преобразования.

Упражнение. Найдите другие константы и сравните получаемый результат.

1.7 Отрисовка кривых

Помимо текста, OpenCv позволяет отрисовать на изображении и стандартные кривые: линия (отрезок), окружность, эллипс, прямоугольник и многоугольник (poly). а также их дуги.

Точка Для отрисовки перечисленных выше примитивов необходимо указание, в частности, точки(ек). Точки задаются как пара чисел:

```
1 Point( 50, 70);//Задана точка с координатами 50, 70.
```

Конструктор Первое число задает координату вдоль оси абсцисс, а второе – вдоль оси ординат. Отмечу, что ось y (ординат) традиционно у изображений направлена вниз. (Важно при выводе изображения на экран.)

Доступ к отдельным компонентам осуществляется так: p.x p.y

Отрезок Для вывода “линии” на изображении используется функция `line`, которая является традиционным названием функции для отрисовки как раз не линий, а фактически отрезков. Исходя из последнего отмеченная функция принимает в качестве аргументов концевые точки отрезка в изображении, цвета и параметры самой линии.

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <string>
using namespace cv;
using namespace std;
int main(int argc, char *argv[])
{
  Mat img(300, 300, CV_8U);
  img *= 0;

  line( img, Point(50, 90), Point(300, 400),
       Scalar(50, 100, 200) );

  imshow( "Display window", img );
  waitKey(0);
  return 0;
}
```

Первые два аргумента задают концевые координаты линии (отрезка). Замечу, что одна из точек лежит вне изображения. Тем не менее отрезок отрисован путем отсечения тех частей которые оказались вне изображения. В более общем случае, как было показано выше, область отсечения можно явно задавать самим `Mat`.

Таким образом получим код: ...

Третий аргумент задает цвет линии.

Толщина линии У данной функции (как и у других из данной серии, в частности, PutText) ещё есть и неявный параметр (в данном случае, четвертый) задающий толщину линии.

```
line( img, Point(50, 90), Point(300, 400),
       Scalar(50, 100, 200), 4 );
```

Толщина измеряется в количестве пикселей.

Окружность Для вывода окружности на изображении используется функция `circle`, которая принимает в качестве аргументов: центр окружности, радиус и цвет.

```
circle( img, Point(100, 150), 70, Scalar(50, 100, 200) );
```

В данном случае будет нарисована окружность с центром в точке $(100, 150)$ радиуса в 70 пикселей. Цвет окружности будет задан тройкой чисел $(50, 100, 200)$.

По аналогии со схожими методами Четвертый аргумент задает толщину линии. Но более интересным является то, что отрицательная толщина обозначает закраску внутренности окружности, т.е. будет нарисован круг.

rectangle FILLED

Эллипс

fillpoly

1.8 Случайные числа

Отрисовка объектов может быть использована для корректного представления результатов работы метода. Например, для обведения найденного объекта или при интерактивном выделении искомых объектов на изображении.

Но помимо прямого назначения отрисовка объектов позволяет формировать сами тестовые изображения. Последнее означает, что для проверки работоспособности некого метода, можно взять фото из жизни, а можно для него синтезировать тестовые изображения (в машинном обучении данное подход превращается в аугментации). Например, для проверки метода поиска окружностей можно создать много изображений на которых нарисованы окружности.

Для упрощения отмеченной выше процедуру потребуются случайные числа. Они конечно существуют отдельно как в стандартной библиотеки языка C, так и в C++. Но лучше использовать версию предоставляемую самим опенсв. Данный объект обозначен как `RNG`.

Для формирования выборки из нормального распределения используется метод `gaussian` данного объекта, аргумент которой задает среднее отклонение Гауссовой случайной величины.

```
1 // Инициализируем геренартор случайных чисел:
2 RNG rng(0x1234);
3 cout << rng.gaussian(0.5) << endl;
```

Конструктор данного объекта позволяет задать первоначальное значение переменной статуса. Такая возможность позволяет иметь повторяемый результат от запуска к запуску. Так, в качестве ответа всегда должно получится число 0.000148816.

Помимо `gaussian` у класса `RNG` есть и метод `uniform`, который генерирует равномерную случайную величину.

Упражнение. Напишите программу рисующая отрезки имеющие случайное положение концов внутри изображения.

Упражнение. Хранитель экрана.

Помимо данной функции имеется и вспомогательная, которая позволяет заполнить все изображение (в общем случае, всю матрицу) случайным числами.

```
1 Mat noise = img.clone();// Формируем копию изображения.
2 RNG rng(0x1234);
3 // Заполняем матрицу/изображение гауссовым шумом:
4 rng.fill( noise, RNG::NORMAL, 0, 50);
```

```
1 img + noise; // Добавляем к изображению шум.
```

randu??

Упражнение. Превратите эти куски кода в программу, которая к входному изображению добавляет шум и показывает результирующее изображение в окошке.