OpenCV C++

Basics

By James Rogers

James.rogers@plymouth.ac.uk

Introduction

The biggest problem with learning OpenCV is knowing where to start. The library is very extensive with hundrards of new functions, classes, and data types. This document will go over a few of the most important parts of OpenCV, to get you started on your project.

I have classed the different parts of this document under three labels:

Essential (E)	Highly recommend reading all content, these functions will be used extensivly.
Useful (U)	Find time to look these over, these tools are very helpful to know about and most will be used.
Nice to Know (N)	More specific functions but good to know about, read if you're intrested.

Contence

Data Types/Classes:

- **Mat (E):** The most important class in OpenCV, used to store images. It acts like an array of pixels, where the user defines the type of pixel to use:
    - **Vec3b (U):** 24-bit full colour pixel, has three 8-bit channels for Red, Green, and Blue.
    - **uchar (U):** 8-bit grey pixel, with only one brightness channel.
    - **ushort (N):** 16-bit grey pixel, allows for more detailed colour information.

  The Mat class also contains useful functions, which can greatly reduce the compexity of your code:

    - **at() (E):** Accesses a single pixel in a Mat for reading/writing.
    - **size() (U):** Returns the size of the Mat.
    - **copyTo() (U):** Copys the Mat to another.

- **Point (E):** Stores a 2D point with an (x, y) value, many functions require Point inputs to define locations on a Mat.
- **Scalar (E):** Stores a pixel/colour value. Functions that draw shapes require a Scalar input to define the colour.
- **Rect (U):** Defines a rectangle, commonly used for accessing regions of interest within a Mat.
- **Size (U):** Stores a width and height value, useful for storing Mat sizes.
- **VideoCapture (U):** This is used to open video streams such as a video, camera, or IP webstream.

Image Functions:

- **imread() (E):** Reads an image from a flie path, and loads it into a Mat.
- **imwrite() (U):** Saves an image to a file path from a Mat.
- **imshow() (E):** Displays a Mat in a window.
- **waitKey() (E):** Captures a keypress within a timeout period, also useful for delays.
- **cvCreateTrackbar() (N):** Creates a slider on a window, good for chaning variables during runtime.

Drawing Functions:

- **circle() (U):** Draws a circle on a Mat.
- **line() (U):** Draws a line on a Mat.
- **rectangle() (U):** Draws a rectangle on a Mat.
- **putText() (N):** Draws text on a Mat.

Maniupulation Functions:

- **cvtColor() (U):** Changes a Mat from one format to another, such as colour to greyscale.
- **resize() (U):** Resizes a Mat to a different size.
- **normalize() (N):** Adjusts pixel values to force the brightest and darkest pixels to be certain values.
- **minMaxLoc() (N):** Finds the maximum and minimum pixel values/positions.
- **blur() (N):** Applies a blur to a Mat.
- **GaussianBlur() (N):** Applies a gaussian blur to a Mat.
- **inRange() (U):** Checks every pixel against some upper and lower limit.

Data Types/Classes

Mat (E)

Mat is the most used class in openCV, allowing you to store and minipulate images. Creating a blank Mat can be done in the following ways:

```
Mat MatName(Height, Width, PixelType, Initial Value);

Mat MatName(Size, PixelType, Initial Value);
```

Height/Width: An interger value defineing the size of your Mat.
Size: Size is a datatype that stores both height and width.
PixelType: A code defining what kind of Mat you want to create:

- **CV_8U**: An 8-bit greyscale image, using **uchar** pixels.
- **CV_8UC3**: An 8-bit 3 channel colour image (RGB), using **Vec3b** pixels.
- **CV_16U**: A 16-bit greyscale image, using **ushort** pixels.
- **More**: A complete list can be found here.

Initial Value: The defult value for the pixels, typically a **Scalar** colour value.

For example, creating a Mat which is 1000 by 500, full colour, and red **(note that colour is written as BGR)**:

```
Mat Mat1(500, 1000, CV_8UC3, Scalar(0,0,255));
```

Or a black greyscale image the same size as the above Mat:

```
Mat Mat2(Mat1.size(), CV_8U, Scalar(0));
```

See imRead() and VideoCapture for importing images from other sourses into a Mat.

at() (E)

This function is useful for accessing a single pixel, to either read it or override it:

```
MatName.at<PixelType>(y,x);
```

Note that the cordinates are in the order y,x. Remember this because it is easy to mix these up. If we were to read the pixel at (100, 200), it can be done like so (where Vec3b is the pixel type):

```
Vec3b PixVal = Mat2.at<Vec3b>(200,100);
```

Pixels can be edited in the same way, just make sure to cast your new value as the Mat pixel type:

```
Mat2.at<Vec3b>(200,100) = (Vec3b)PixVal;
```

size() (U):

Returns the size of a Mat, but can also be broken down into width, height, and area. This is useful for nested for-loops which access each pixel value, regardless of the size of the Mat:

```
for(int y=0; y<Mat1.size().height; y++){
    for(int x=0; x<Mat1.size().width; x++){
        //Pixel level code here
    }
}
```

CopyTo() (U):

One problem with Mats is that they cannot simply be copied like a variable:

```
Mat Mat2=Mat1;
```

Doing so will create a Mat that can be used, but they share the same memory space, thus edits to Mat2 will apply to Mat1 and vice versa. To make a copy that can be independently edited without linking to the orginal, the CopyTo command is used:

```
Mat Mat2;
Mat1.copyTo(Mat2);
```

Point (E)

A Point is used to store an (x, y) value, typically to define a position on a Mat:

```
Point myPoint(240,480);
```

Points can be manipulated in a number of ways:

```
Point3=Point1+Point2; //Element by element Addition
      Point1=Point1*5; //Scaling each element
       Point1.x=120; //Element level editing
```

Scalar (E)

A Scalar is nice general purpose data type for colour. It can be used to store single channel grey values, RGB colour values, or even RGBA 4 channel values. Below are some example colours:

- Single Channel white: `Scalar myColour(255);`
- Three Channel white: `Scalar myColour(255,255,255);`
- Red: `Scalar myColour(0,0,255);`
- Green: `Scalar myColour(0,255,0);`
- Blue: `Scalar myColour(255,0,0);`

Rect (U)

Rectangles are useful for defining areas within Mats, for example a face or object. They are defined by the top left corner position, width, and height. There are a few different ways of defineing a Rect depending on what information you have:

```
Rect myRect(x, y, w, h); //Rect with top left at (x,y), with size w*h

Rect myRect(Point_Top_Left, Point_Bottom_Right); //Rect between two points

Rect myRect(Point_Top_Left, RectSize); //Rect at a Point, with a size
```

Rects can be used with Mats to crop areas of interest. For example, making a copy of a target region:

```
Rect target(100,200,50,50); //Create a rectangle at (100,200) with size 50*50
Mat targetImage; //Initialize an empty Mat
CameraFrame(target).copyTo(targetImage); //Copy just the target region
```

Size (U)

Simular to Point, Size stores two values regarding the x and y axis, but is more specialised for height and width. Useful for storing size information about Mats:

```
Size Size1 = Mat1.size();
```

VideoCapture (U)

This is a class used for opening and captruing frames from a video source, such as a camera or IP stream. Every camera connected to a computer has an ID number, starting at 0 and incrementing with more devices that are connected. To create a videoCapture instance, and connect it to camera 0, use the “open” commad:

```
VideoCapture Cap;
Cap.open(0);
```

It is always a good idea to check that the camera has beed sucsefully opened before trying to use it. This can be done with the “isOpened” function:

```
if(!Cap.isOpened()){
    cout<<"Error opening camera"<<endl;
}
```

To grab frames from the camera, create an empty Mat and use the “read” function to write to it:

```
Mat Frame;
Cap.read(Frame);
```

Image Functions

imread() (E)

This function allows you to read image files from a file path and load them into a Mat. Supported image formats are as followes:

- Windows bitmaps - *.bmp, *.dib
- JPEG files - *.jpeg, *.jpg, *.jpe
- JPEG 2000 files - *.jp2
- Portable Network Graphics - *.png
- WebP - *.webp
- Portable image format - *.pbm, *.pgm, *.ppm *.pxm, *.pnm
- Sun rasters - *.sr, *.ras
- TIFF files - *.tiff, *.tif
- OpenEXR Image files - *.exr
- Radiance HDR - *.hdr, *.pic
- Raster and Vector geospatial data supported by GDAL

```
Mat1 = imread("C:/Repository/Projects/myProject/image1.jpg");
```

By defualt, images are loaded in as an 8-bit 3 channel image (CV_8UC3), even if the source image is greyscale.

imwrite() (U)

Simular to imread, but saves a Mat to a file path. The output image type is based on the file extension, and follows the same compatibility list as above (imread).

```
imwrite("C:/Repository/Projects/myProject/image1.jpg", Mat1);
```

Note, if the file path already exists, it will be overriden.

imshow() (E)

Dispays a Mat to the screen within a window. However for this to work you **NEED** a waitKey(10) after using it, as it introduces a small delay long enough to diplay the image. It is only required once per frame, so multible imshow() calls can happen before just one waitKey(10) call (each window must have a uniqe name, otherwise it will be overriden):

```
imshow("WindowName1", Mat1);
imshow("WindowName2", Mat2);
imshow("WindowName3", Mat3);
waitKey(10);
```

waitKey() (E)

This is used to wait for and grab a keypress, but is also useful as a delay. It has one input, which is the time out duration in miliseconds. The function will wait said time, and if a key is pressed within it, the keys ascii character will be returned. (entering a duration of 0 will make it block untill a key is pressed).

Typical applications use this function within a switch-case statement, but can be used in a number of ways:

Example 1:

```
switch(waitKey(10)){
    case 'a':
        //Do something when 'a' is pressed
        break;
    case 'b':
        //Do something when 'b' is pressed
        break;
}
```

Example 2:

```
if(waitKey(10)=='a'){
    //Do something when 'a' is pressed
}
```

***NOTE: Key presses will only be registered if an openCV window is focused (see "imshow()").**

cvCreateTrackbar() (N)

Allows live manipulation of variables via a slide bar. The function takes a window name, and adds a slider to the bottom of it. It also needs the address of the variable you wish to modify (only works on intergers). **Make sure the variable has higher scope than the function**, so that the pointer doesn't get dereferanced. An easy way of doing this is to make the variable static or global. The last input of the function is the limit, in the example below, a slider called "SliderName" will be added to a window called "WindowName", with a range of 0 to 100. Moving said slider will modify the value stored in Var.

```
static int Var=0;
cvCreateTrackbar("SliderName", "WindowName", &Var, 100);
```

Drawing Functions

circle() (U)

Draws a circle onto a Mat, at a given location, with a set size, colour, and thickness. In the below example, a circle is drawn on Mat1, at position (100,50), with a radius of 20 pixels, colour green, and a line thickness of 3. If the line thickness is set to -1, the circle is filled in rather than just being an outline.

```
circle(Mat1, Point(100,50), 20, Scalar(0,255,0), 3);
```

line() (U)

Draws a line onto a Mat between two points. The example below draws a line from (20,100) to (300,500), coloured blue and 2 pixels thick.

```
line(Mat1, Point(20,100), Point(300, 500), Scalar(255,0,0), 2);
```

rectangle() (U)

Draws a rectangle onto a Mat, from a Rect, or between two points:

Example 1: Draw a rectangle with its top-left point at (20,10), and a size of 300x400. Make it red and set its line thickness to 4.

```
rectangle(Mat1, Rect(20,10,300,400), Scalar(0,0,255), 4);
```

Example 2: Draw a rectangle with one corner at (800,200) and the other at (400,70). Make it white and filled solid.

```
rectangle(Mat1, Point(800,200), Point(400, 70), Scalar(255,255,255), -1);
```

putText() (N)

Draws text onto a Mat, given: a position, string, font type, text size, colour, and thickness. Font types can be any of the following:

- **FONT_HERSHEY_SIMPLEX** normal size sans-serif font.
- **FONT_HERSHEY_PLAIN** small size sans-serif font.
- **FONT_HERSHEY_DUPLEX** normal size sans-serif font (more complex than FONT_HERSHEY_SIMPLEX).
- **FONT_HERSHEY_COMPLEX** normal size serif font.
- **FONT_HERSHEY_TRIPLEX** normal size serif font (more complex than FONT_HERSHEY_COMPLEX).
- **FONT_HERSHEY_COMPLEX_SMALL** smaller version of FONT_HERSHEY_COMPLEX.
- **FONT_HERSHEY_SCRIPT_SIMPLEX** hand-writing style font.
- **FONT_HERSHEY_SCRIPT_COMPLEX** more complex variant of FONT_HERSHEY_SCRIPT_SIMPLEX.

Note: Marks will be deducted if you find a way to add comic sans to OpenCV.

The below example will place text with its top-left corner at (100,200), with font type Hershey Simplex, size 2, coloured black, and line thickness 3.

```
putText(Mat1,"Hello World!",Point(100,200),FONT_HERSHEY_SIMPLEX,2.0,Scalar(0,0,0),3);
```

Maniupulation Functions

cvtColor() (U)

Converts one image type to another, for example, colour to greyscale. The type of conversion done is based on the conversion code entered. In the following example, Mat1 is converted from a BGR 3-channel image to a grey single channel image.

```
cvtColor(Mat1, Mat1, CV_BGR2GRAY);
```

For a complete list of conversion codes, look here. **Note that some versions of OpenCV use the COLOR prefix instead of CV. So CV_BGR2GRAY becomes COLOR _BGR2GRAY.**

resize() (U)

Resize can upscale or downscale a Mat to a new size. It takes an input Mat, an output Mat, and a Size. The below example takes Mat1, resizes it to 400 x 200, and stores it in Mat2. Note the since Mat1 is an input Mat, it is not modified by this code.

```
resize(Mat1,Mat2,Size(400,200));
```

In the next example, Mat2 is resized to be the same size as Mat3. Only Mat2 is modified in this line.

```
resize(Mat2,Mat2,Mat3.size());
```

normalize() (N)

Normalize essentially increases contrast in an image. It finds the minimum and maximum values in a Mat, and sets them to new values, as well as rescaling all the levels in between. For example, if you had a greyscale image, and you wanted the darkest colour to be pitch black, and the lightest colour to be pure white, then use the following line:

```
normalize(Mat1, Mat1, 0, 255, CV_MINMAX, CV_8U);
```

Mat1 is the input image, Mat1 is also the output image (hence writing it twice), the new min value is 0, and the new max value is 255. CV_MINMAX is the normalization type, and CV_8U is the pixel type of Mat1 (see the Mat section for details). **Note that some versions of OpenCV use NORM_MINMAX instead of CV_MINMAX.**

minMaxLoc() (N)

This function returns the position and value of the maximum and minimum pixels on a single channel image. This is a necessary step in cross corrilation for example, where the best match is the brightest pixel on the template match output. One quirk with this function, is that if you just want to find the max or min value, you have to find both, even if the other isn't used.

Variables must be predefined to store the outputs, which are 2 Points for the min and max positions, and 2 doubles for the min and max values. The code below finds the min and max positions/values for Mat1.

```
double minVal,maxVal;
Point minLoc, maxLoc;

minMaxLoc(Mat1, &minVal, &maxVal, &minLoc, &maxLoc);
```

blur() (N)

Applies a simple average blur over the whole image. This is done by replacing each pixel with an average of its surroundings. The number of surrounding pixels used in this calculation defines the strength of the blur. The code below takes Mat1, applies and 11x11 blur, and stores the result back into Mat1. An 11x11 blur means that each pixel is replaced with the average of an 11x11 pixel box around it.

```
blur(Mat1, Mat1, Size(11,11));
```

As this blur evaluates surrounding pixels equally, and surrounding pixels are defined within a sqaure, this blur method can produce some artfacts on strong contrasting boundries. However, it runs a lot faster than GaussianBlur(), which produces a more natural result.

GaussianBlur() (N)

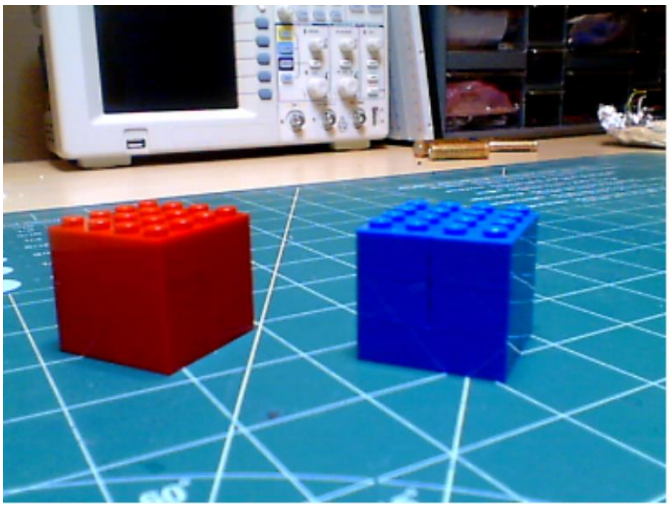
A gaussian blur is more computentially expensive than a simple mean blur, however produces a more natural result, much like defocusing a camera. Each pixel is replaced with the average of surrounding pixels, each weighted based on a gaussian distribution.

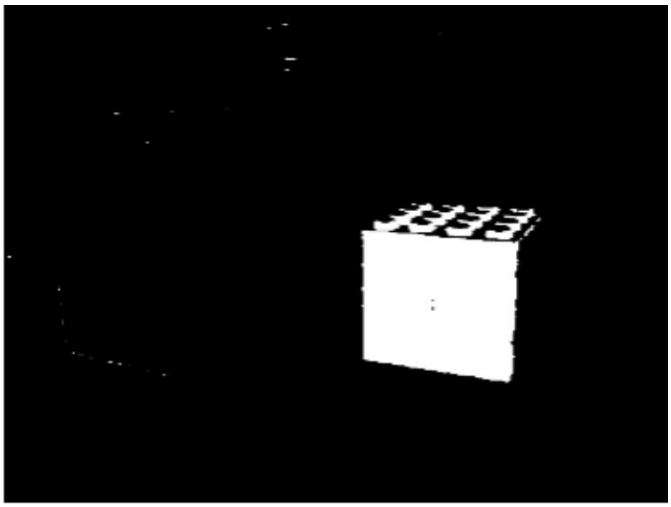
```
GaussianBlur(Mat1, Mat1, Size(11,11), 0);
```

The code above takes Mat1, applies an 11x11 gaussian blur, and stores the result in Mat1.

inRange() (U)

This function is used for thresholding an image against some value or range. For example in colour tracking, you can set a range of colours to be the target colour. The output image will return white pixels for positive detections and black pixels for negative detections. In the example below, the colour image was processed by inRange() for dark blue colours:





```
inRange(inputMat, Vec3b(120,100,100), Vec3b(130,255,255), outputMat);
```

This example was processed in the HSV colour space, so the Vec3b limits in the above function represent hue/saturation/value. The first limit omits any pixels with a hue less that 120, a saturation less than 100, and a value of less than 100. The second limit omits any pixels with a hue greater than 130, a saturation greater than 255, and a value greater than 255.