**DOCUMENT ABOUT CVTCOLOR METHOD AND CONVERT A GRAY IMAGE TO A BINARY IMAGE**

1. **cvtColor method**

Converts an image from one color space to another.

**C++:** void cvtColor(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn**=0 )

|  |  |
| --- | --- |
| **Parameters:** | * **src** – input image: 8-bit unsigned, 16-bit unsigned ( CV\_16UC... ), or single-precision floating-point. * **dst** – output image of the same size and depth as src. * **code** – color space conversion code (see the description below). * **dstCn** – number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from src and code . |

The function converts an input image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on.

The conventional ranges for R, G, and B channel values are:

* 0 to 255 for CV\_8U images
* 0 to 65535 for CV\_16U images
* 0 to 1 for CV\_32F images

In case of linear transformations, the range does not matter. But in case of a non-linear transformation, an input RGB image should be normalized to the proper value range to get the correct results, for example, for RGB \rightarrowL\*u\*v\* transformation. For example, if you have a 32-bit floating-point image directly converted from an 8-bit image without any scaling, then it will have the 0..255 value range instead of 0..1 assumed by the function. So, before calling cvtColor , you need first to scale the image down:

img \*= 1./255;

cvtColor(img, img, CV\_BGR2Luv);

If you use cvtColor with 8-bit images, the conversion will have some information lost. For many applications, this will not be noticeable but it is recommended to use 32-bit images in applications that need the full range of colors or that convert an image before an operation and then convert back.

If conversion adds the alpha channel, its value will set to the maximum of corresponding channel range: 255 for CV\_8U, 65535 for CV\_16U, 1 for CV\_32F.

The function can do the following transformations:

* RGB \leftrightarrowGRAY ( CV\_BGR2GRAY, CV\_RGB2GRAY, CV\_GRAY2BGR, CV\_GRAY2RGB ) Transformations within RGB space like adding/removing the alpha channel, reversing the channel order, conversion to/from 16-bit RGB color (R5:G6:B5 or R5:G5:B5), as well as conversion to/from grayscale using:

\text{RGB[A] to Gray:} \quad Y  \leftarrow 0.299  \cdot R + 0.587  \cdot G + 0.114  \cdot B

and

\text{Gray to RGB[A]:} \quad R  \leftarrow Y, G  \leftarrow Y, B  \leftarrow Y, A  \leftarrow \max (ChannelRange)

The conversion from a RGB image to gray is done with:

cvtColor(src, bwsrc, CV\_RGB2GRAY);

More advanced channel reordering can also be done with [mixChannels()](http://docs.opencv.org/modules/core/doc/operations_on_arrays.html#void mixChannels(const Mat* src, size_t nsrcs, Mat* dst, size_t ndsts, const int* fromTo, size_t npairs)) .

* RGB \leftrightarrowCIE XYZ.Rec 709 with D65 white point ( CV\_BGR2XYZ, CV\_RGB2XYZ, CV\_XYZ2BGR, CV\_XYZ2RGB ):

\begin{bmatrix} X  \\ Y  \\ Z
  \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227
  \end{bmatrix} \cdot \begin{bmatrix} R  \\ G  \\ B
  \end{bmatrix}

\begin{bmatrix} R  \\ G  \\ B
  \end{bmatrix} \leftarrow \begin{bmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 &  1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311
  \end{bmatrix} \cdot \begin{bmatrix} X  \\ Y  \\ Z
  \end{bmatrix}

X, Yand Zcover the whole value range (in case of floating-point images, Zmay exceed 1).

* RGB \leftrightarrowYCrCb JPEG (or YCC) ( CV\_BGR2YCrCb, CV\_RGB2YCrCb, CV\_YCrCb2BGR, CV\_YCrCb2RGB )

Y  \leftarrow 0.299  \cdot R + 0.587  \cdot G + 0.114  \cdot B

Cr  \leftarrow (R-Y)  \cdot 0.713 + delta

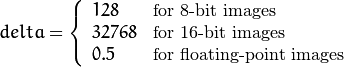
Cb  \leftarrow (B-Y)  \cdot 0.564 + delta

R  \leftarrow Y + 1.403  \cdot (Cr - delta)

G  \leftarrow Y - 0.714  \cdot (Cr - delta) - 0.344  \cdot (Cb - delta)

B  \leftarrow Y + 1.773  \cdot (Cb - delta)

where



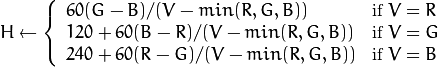
Y, Cr, and Cb cover the whole value range.

* RGB \leftrightarrowHSV ( CV\_BGR2HSV, CV\_RGB2HSV, CV\_HSV2BGR, CV\_HSV2RGB )

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

V  \leftarrow max(R,G,B)

S  \leftarrow \fork{\frac{V-min(R,G,B)}{V}}{if $V \neq 0$}{0}{otherwise}



If H<0then H \leftarrow H+360. On output 0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360.

The values are then converted to the destination data type:

* + 8-bit images

V  \leftarrow 255 V, S  \leftarrow 255 S, H  \leftarrow H/2  \text{(to fit to 0 to 255)}

* + 16-bit images (currently not supported)

V <- 65535 V, S <- 65535 S, H <- H

* + 32-bit images

H, S, and V are left as is

* RGB \leftrightarrowHLS ( CV\_BGR2HLS, CV\_RGB2HLS, CV\_HLS2BGR, CV\_HLS2RGB ).

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

V_{max}  \leftarrow {max}(R,G,B)

V_{min}  \leftarrow {min}(R,G,B)

L  \leftarrow \frac{V_{max} + V_{min}}{2}

S  \leftarrow \fork { \frac{V_{max} - V_{min}}{V_{max} + V_{min}} }{if  $L < 0.5$ }
    { \frac{V_{max} - V_{min}}{2 - (V_{max} + V_{min})} }{if  $L \ge 0.5$ }

H  \leftarrow \forkthree {{60(G - B)}/{S}}{if  $V_{max}=R$ }
  {{120+60(B - R)}/{S}}{if  $V_{max}=G$ }
  {{240+60(R - G)}/{S}}{if  $V_{max}=B$ }

If H<0then H \leftarrow H+360. On output 0 \leq L \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360.

The values are then converted to the destination data type:

* + 8-bit images

V  \leftarrow 255 \cdot V, S  \leftarrow 255 \cdot S, H  \leftarrow H/2 \; \text{(to fit to 0 to 255)}

* + 16-bit images (currently not supported)

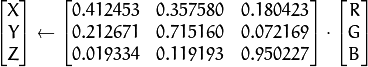
V <- 65535 \cdot V, S <- 65535 \cdot S, H <- H

* + 32-bit images

H, S, V are left as is

* RGB \leftrightarrowCIE L\*a\*b\* ( CV\_BGR2Lab, CV\_RGB2Lab, CV\_Lab2BGR, CV\_Lab2RGB ).

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.



X  \leftarrow X/X_n,  \text{where} X_n = 0.950456

Z  \leftarrow Z/Z_n,  \text{where} Z_n = 1.088754

L  \leftarrow \fork{116*Y^{1/3}-16}{for $Y>0.008856$}{903.3*Y}{for $Y \le 0.008856$}

a  \leftarrow 500 (f(X)-f(Y)) + delta

b  \leftarrow 200 (f(Y)-f(Z)) + delta

where

f(t)= \fork{t^{1/3}}{for $t>0.008856$}{7.787 t+16/116}{for $t\leq 0.008856$}

and

delta =  \fork{128}{for 8-bit images}{0}{for floating-point images}

This outputs 0 \leq L \leq 100, -127 \leq a \leq 127, -127 \leq b \leq 127. The values are then converted to the destination data type:

* + 8-bit images

L  \leftarrow L*255/100, \; a  \leftarrow a + 128, \; b  \leftarrow b + 128

* + 16-bit images

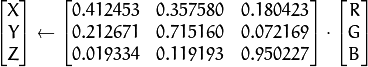
(currently not supported)

* + 32-bit images

L, a, and b are left as is

* RGB \leftrightarrowCIE L\*u\*v\* ( CV\_BGR2Luv, CV\_RGB2Luv, CV\_Luv2BGR, CV\_Luv2RGB ).

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit 0 to 1 range.



L  \leftarrow \fork{116 Y^{1/3}}{for $Y>0.008856$}{903.3 Y}{for $Y\leq 0.008856$}

u'  \leftarrow 4*X/(X + 15*Y + 3 Z)

v'  \leftarrow 9*Y/(X + 15*Y + 3 Z)

u  \leftarrow 13*L*(u' - u_n)  \quad \text{where} \quad u_n=0.19793943

v  \leftarrow 13*L*(v' - v_n)  \quad \text{where} \quad v_n=0.46831096

This outputs 0 \leq L \leq 100, -134 \leq u \leq 220, -140 \leq v \leq 122.

The values are then converted to the destination data type:

* + 8-bit images

L  \leftarrow 255/100 L, \; u  \leftarrow 255/354 (u + 134), \; v  \leftarrow 255/262 (v + 140)

* + 16-bit images

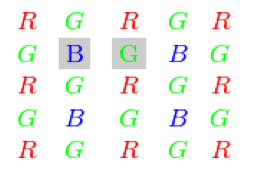
(currently not supported)

* + 32-bit images

L, u, and v are left as is

The above formulae for converting RGB to/from various color spaces have been taken from multiple sources on the web, primarily from the Charles Poynton site <http://www.poynton.com/ColorFAQ.html>

* Bayer \rightarrowRGB ( CV\_BayerBG2BGR, CV\_BayerGB2BGR, CV\_BayerRG2BGR, CV\_BayerGR2BGR, CV\_BayerBG2RGB, CV\_BayerGB2RGB, CV\_BayerRG2RGB, CV\_BayerGR2RGB ). The Bayer pattern is widely used in CCD and CMOS cameras. It enables you to get color pictures from a single plane where R,G, and B pixels (sensors of a particular component) are interleaved as follows:



The output RGB components of a pixel are interpolated from 1, 2, or 4 neighbors of the pixel having the same color. There are several modifications of the above pattern that can be achieved by shifting the pattern one pixel left and/or one pixel up. The two letters C_1and C_2in the conversion constants CV\_Bayer C_1 C_22BGR and CV\_Bayer C_1 C_22RGB indicate the particular pattern type. These are components from the second row, second and third columns, respectively. For example, the above pattern has a very popular “BG” type.

**\* Code demo:**

How to use cvtColor in OpenCV



Result:



1. **How to converts gray image to binage image**
2. We can use “static threshold”:

- First, we can choose a gray image.

- After that, we choose a static threshold for this image.

**\* Advantage**

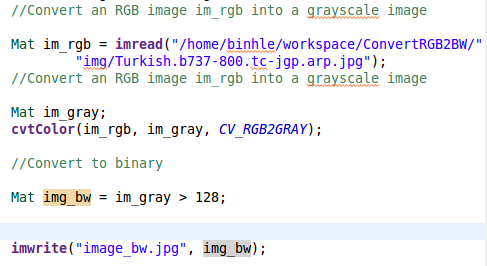
- This method is easy to use, easy to understand.

- Coding easy.

**\* Disadvantage**

- We choose a static threshold so we must focus about bright of image.

**\* Code demo:**

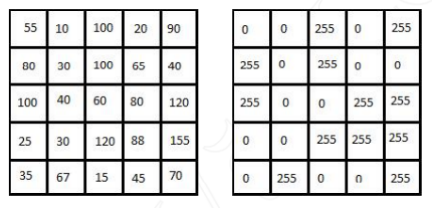


**\* Result**



1. We can use “Dymamic threshold” adaptiveThreshold method in OpenCV

This method will cut us gray image to small window. For example:



The binary image is obtained on the binary image with threshold is the avaerage of all the cells in window T = (55 +10 + 100 + …)/25 = 65.5.

**\* Advantage**

- Cannot focus about bright of image.

**\* Disadvantage**

- Must know about OpenCV and using adaptiveThreshold method.

**\* AdaptiveThreshold method:**

Applies an adaptive threshold to an array.

C++: void adaptiveThreshold(InputArray src, OutputArray dst, double maxValue, int adaptiveMethod, int thresholdType, int blockSize, double C)

|  |  |
| --- | --- |
| Parameters: | * src – Source 8-bit single-channel image. * dst – Destination image of the same size and the same type as src . * maxValue – Non-zero value assigned to the pixels for which the condition is satisfied. See the details below. * adaptiveMethod – Adaptive thresholding algorithm to use, ADAPTIVE\_THRESH\_MEAN\_C or ADAPTIVE\_THRESH\_GAUSSIAN\_C . See the details below. * thresholdType – Thresholding type that must be either THRESH\_BINARY or THRESH\_BINARY\_INV . * blockSize – Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on. * C – Constant subtracted from the mean or weighted mean (see the details below). Normally, it is positive but may be zero or negative as well. |

The function transforms a grayscale image to a binary image according to the formulae:

* THRESH\_BINARY



* THRESH\_BINARY\_INV



where  is a threshold calculated individually for each pixel.

* adaptiveMethod: 2 type is **ADAPTIVE\_THRESH\_MEAN\_C** and **ADAPTIVE\_THRESH\_GAUSSIAN\_C**
* **ADAPTIVE\_THRESH\_MEAN\_C :** threshold value is the mean of neighbourhood area.

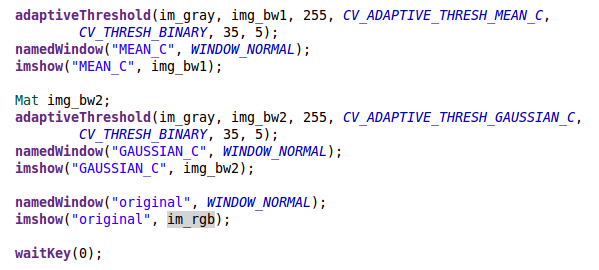


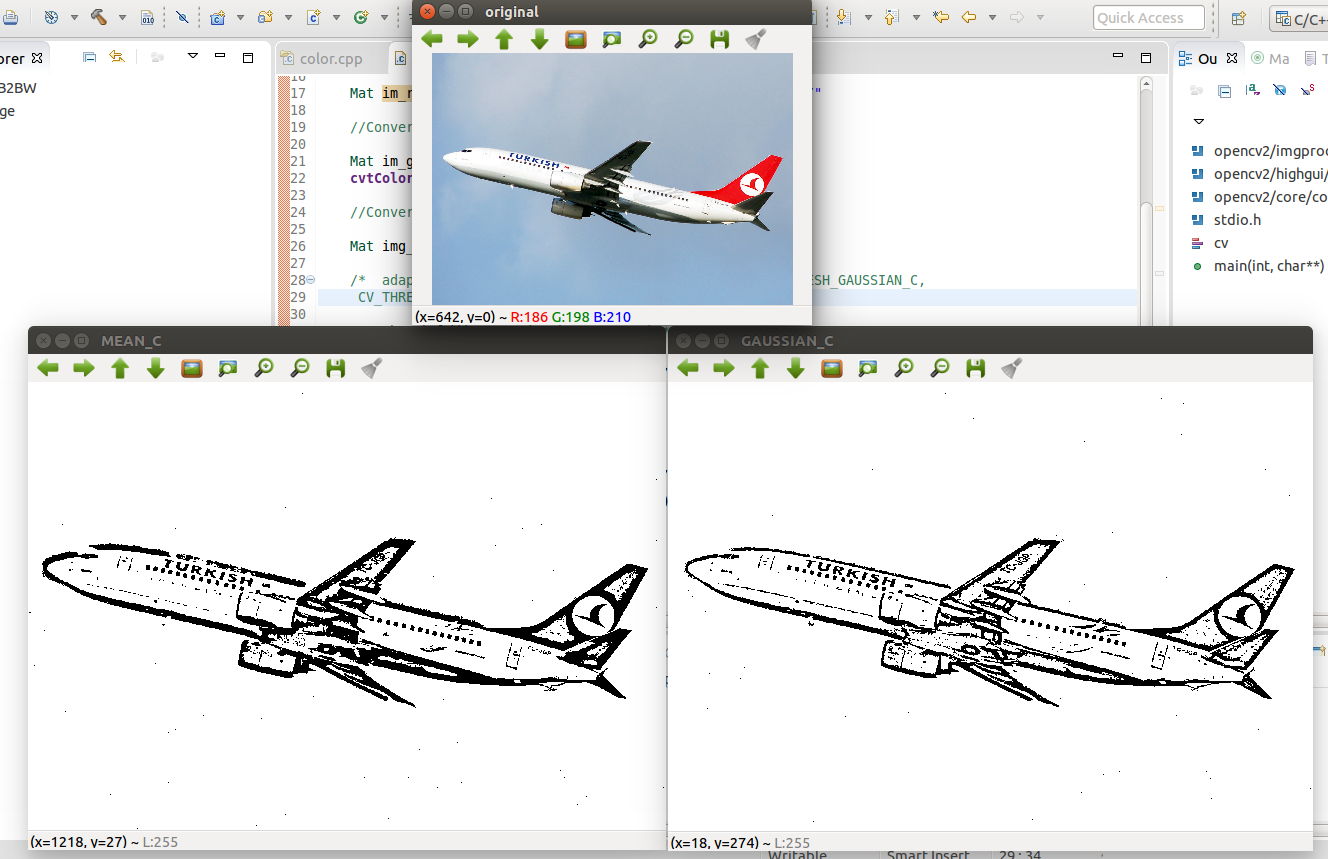
+ ADAPTIVE\_THRESH\_MEAN\_C use for we want to have a binary detail of image but this image have more image noise.

* **ADAPTIVE\_THRESH\_GAUSSIAN\_C:** threshold value is the weighted sum of neighbourhood values where weights are a gaussian window.

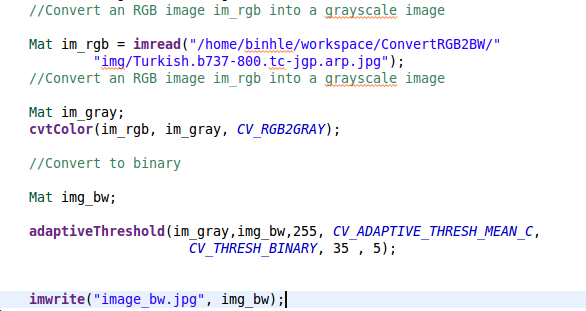


+ ADAPTIVE\_THRESH\_GAUSSIAN\_C use for we want to have image less image noise, but this image left more detail of original image.

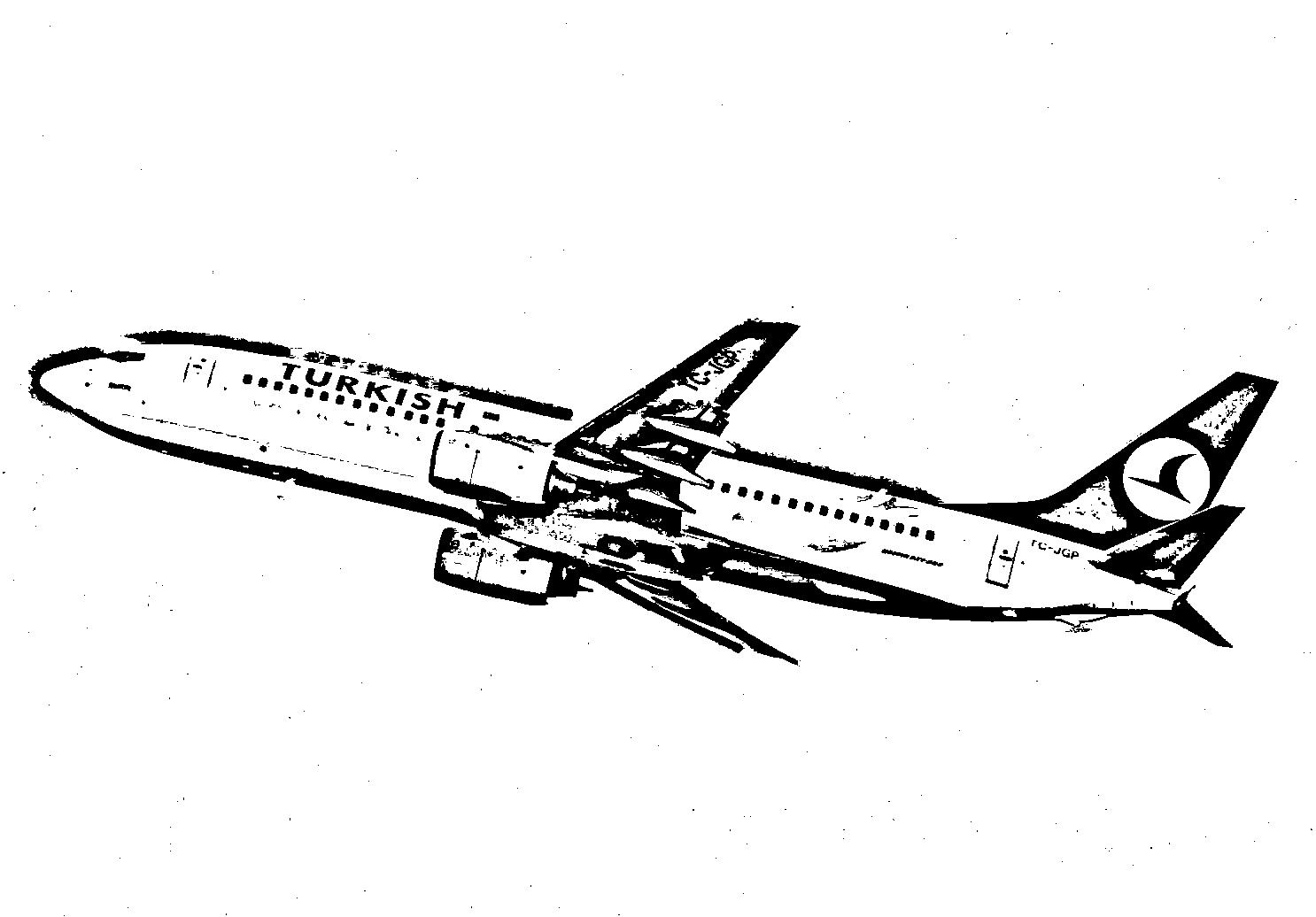
\* Demo different between **ADAPTIVE\_THRESH\_MEAN\_C** and **ADAPTIVE\_THRESH\_GAUSSIAN\_C**



**\* Code Demo converts a gray image to a binary image**



**\* Result**



**Finally**, that reasons why we choose convert a gray image to a binary image by **AdaptiveThreshold** with adaptiveMethod is **ADAPTIVE\_THRESH\_GAUSSIAN\_C.** Because this project needs a binary image clear without image noise.

Using:

OpenCV Document

Stackoverflow.com

Image demo in Google.com