OpenCV Image beautification

1.Color picture histogram

The color histogram refers to the color distribution in an image, which is independent of the specific objects in the image. It is only used to represent the color distribution in the image observed by human eyes, for example, how much proportion of red and green in a picture.

As we know, the computer color display adopts the principle of R, G and B additive color mixing, and emits three electron beams of different intensities to make the red, green and blue phosphorescent materials covered on the inside of the screen emit light to generate color. In the RGB color space, any color light f can be mixed by adding different components of RGB three colors. That is, the color value of each pixel in the image can be represented by a ternary value (R, G, b), for example, (0,0,0) represents black, (0,0255) represents blue,... The size of each component ranges from 0 to 255

Then, we can know that a color image can be stacked by R, G and B channels. It can be imagined that three sheets of paper of the same size are stacked, and then the image seen by the human eye from the top to the bottom is the original color image. Here, assuming the resolution of the image is 320 * 240, then the length and width of each channel are 320 and 240, and the unit is pixels. The total number of pixels is 3 * 320 * 240.

Way 1：

Using the sub library pyplot of Matplotlib, it provides a drawing framework similar to Matlab. Matplotlib is a very powerful Python drawing package. The histogram drawing is mainly implemented by calling hist() function, which draws the histogram according to the data source and pixel level.

hist() :The function form is as follows:：

hist(data source, pixel level)

Wherein, parameters:

Data source: it must be a one-dimensional array, and it is usually necessary to straighten the image through the function ravel()

Pixel level: generally 256, indicating [0, 255]

The function ravel() reduces a multi-dimensional array to a one-dimensional array. The format is:

One dimensional array = multidimensional array. ravel ()

eg: plt.hist(img.ravel(), 256) #ravel() Grouping of two-dimensional reduced one-dimensional 256 gray levels

Way 2：

Opencv provides us with the following functions:

cv2.calcHist(image,channels,mask,histSize,ranges[,hist[,accumulate]])

This function has five parameters:

Image: the input image should be enclosed with square brackets [] when it is passed in

Channels: the channel of the incoming image. If it is a gray-scale image, it is needless to say that there is only one channel with a value of 0. If it is a color image (with 3 channels), the value is 0, 1, 2. Select one of them, corresponding to each channel of BGR. This value must also be passed in with [].

Mask: mask image. If the whole picture is counted, it is none. The main reason is that if we want to count the histograms of some graphs, we must construct the corresponding inflammation mask to calculate.

Histsize: number of gray levels, square brackets are required, such as [256]
Ranges: the range of pixel values, usually [0256]. If some images are not 0-256, for example, if the pixel values are negative and large due to various transformations, they need to be adjusted.

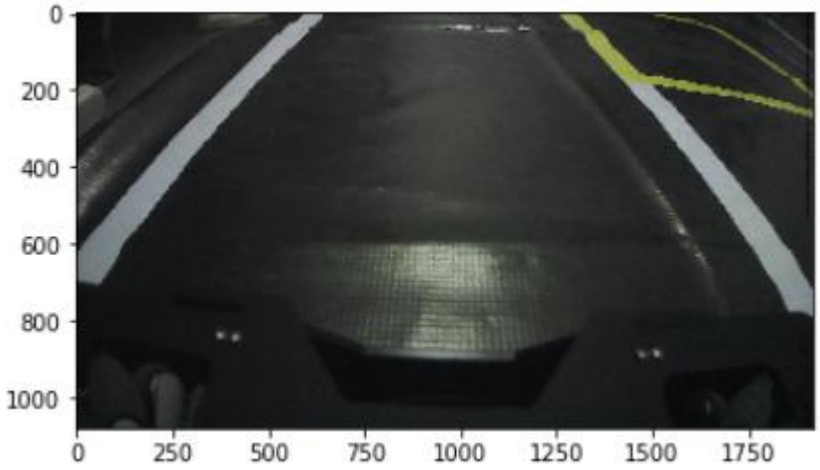
```
import cv2

import numpy as np

import matplotlib.pyplot as plt



img = cv2.imread('Road_test.jpg', 1)

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img)

plt.show()
```

```
# plot Draw histogram
```

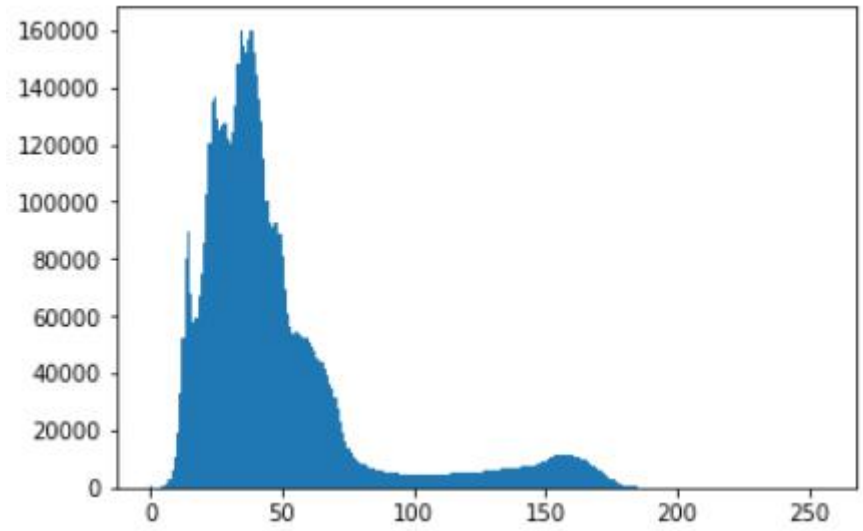
```
plt.hist(img.ravel(), 256)        #ravel() Grouping of two-dimensional reduced one-dimensional 256 gray levels

plt.show()
```

```
# plot    Draw the histogram value returned by opencv
```

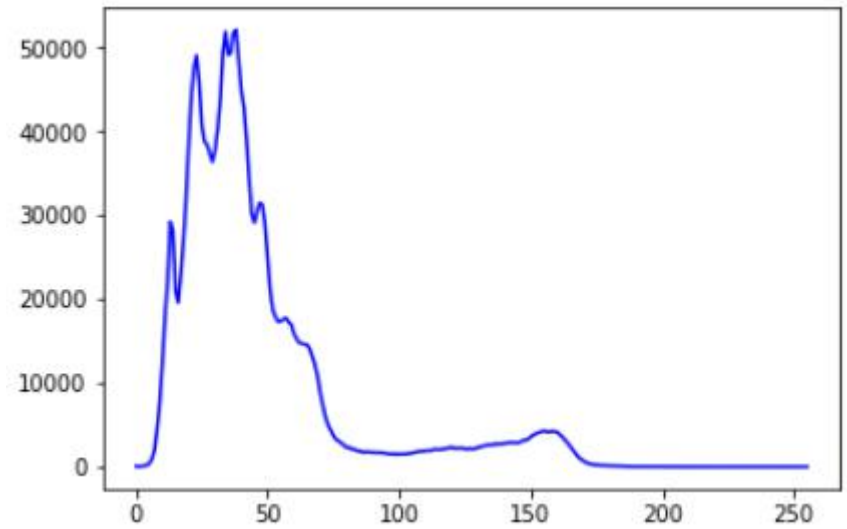
```
histb = cv2.calcHist([img], [0], None, [256], [0, 255])

plt.plot(histb, color='b')

plt.show()
```



Original image



plot Draw histogram

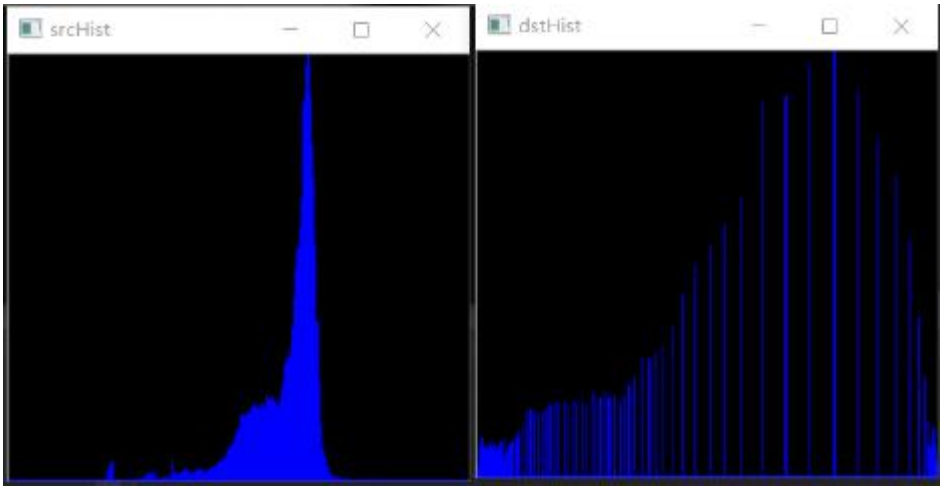


OpenCV Draw histogram

2.Histogram equalization

Image spatial processing is an important image processing technology. This kind of method is directly based on the pixel operation of the image, and is mainly divided into two categories: gray-scale transformation and spatial filtering. Histogram equalization is a commonly used gray-scale transformation method. Generally, the components of the histogram of the dark image are concentrated at the lower end of the gray level, while the components of the histogram of the bright image are biased toward the higher end of the gray level.

If the gray histogram of an image almost covers the whole range of gray values, and the distribution of the whole gray values is nearly uniform except for the number of individual gray values, then the image has a large gray dynamic range and a high contrast, and the details of the image are more abundant. It has been proved that only relying on the histogram information of the input image, a transformation function can be obtained, and the input image can achieve the above effect by using the transformation function. This process is histogram equalization.

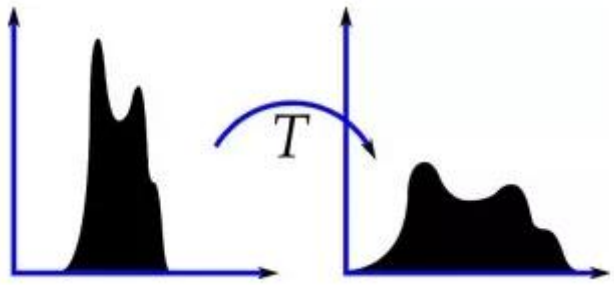


In the figure, the left side is the histogram, and the right side is the histogram equalization effect.

Histogram equalization is to stretch the original histogram to make it evenly distributed in the whole gray range, so as to enhance the contrast of the image.

The central idea of histogram equalization is to change the gray histogram of the original image from a relatively concentrated area to a uniform distribution in the whole gray range. The purpose is to make the overall effect of the image uniform, and the points between each pixel level between black and white are more uniform.

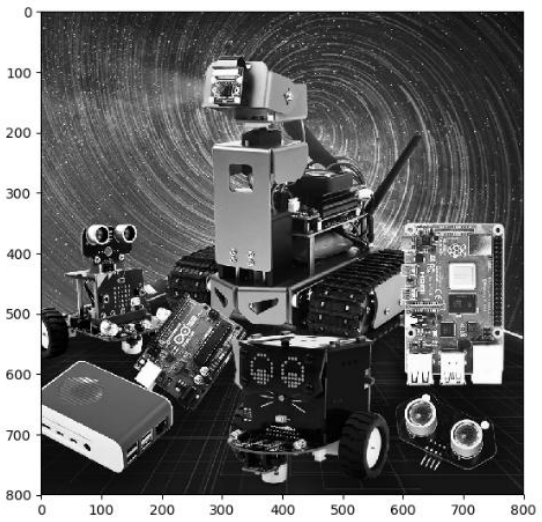
function：cv2.equalizeHist()

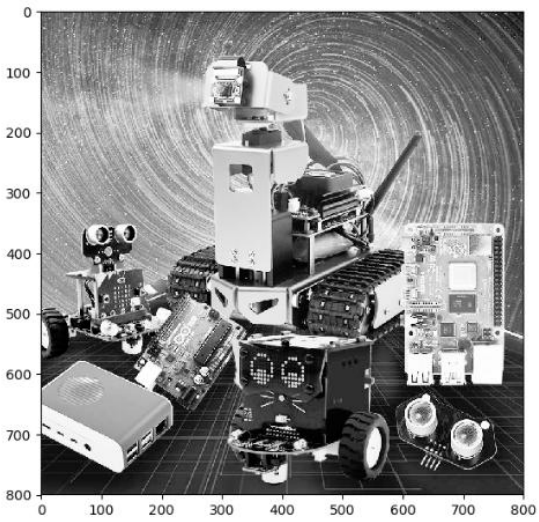



```
# gray Histogram equalization
```

```
import cv2

import numpy as np

import matplotlib.pyplot as plt




img = cv2.imread('yahboom.jpg',1)

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

#cv2.imshow('src',gray)

dst = cv2.equalizeHist(gray)

#cv2.imshow('dst',dst)

#cv2.waitKey(0)




img = cv2.cvtColor(gray, cv2.COLOR_BGR2RGB)

dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)

#plt    Display effect of two pictures before and after drawing

#Source graph display

plt.figure(figsize=(14, 9), dpi=100)#Sets the size and pixels of the drawing area

plt.subplot(121)    # One row, two columns, the first

plt.imshow(img)
```

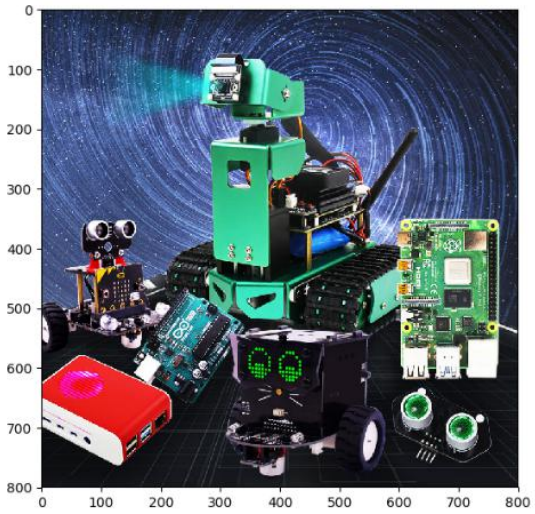
```
#gray Histogram equalization

plt.subplot(122)    # One row, two columns, the second

plt.imshow(dst)

plt.show()
```

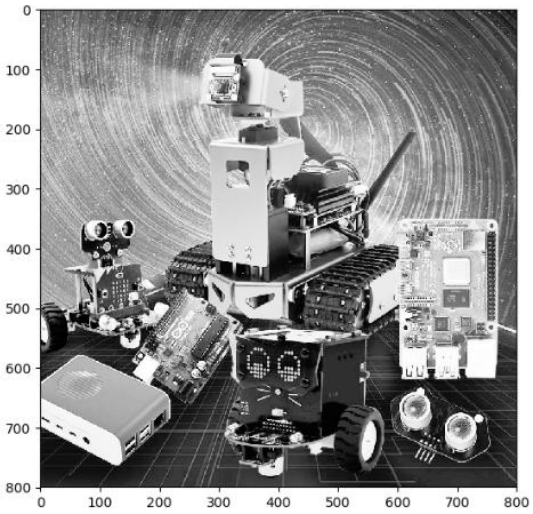





```
# Color histogram equalization

import cv2

import numpy as np

img = cv2.imread('yahboom.jpg',1)

# cv2.imshow('src',img)

(b,g,r) = cv2.split(img)#Channel decomposition

bH = cv2.equalizeHist(b)

gH = cv2.equalizeHist(g)

rH = cv2.equalizeHist(r)

result = cv2.merge((bH,gH,rH))# Channel synthesis

# cv2.imshow('dst',result)
```

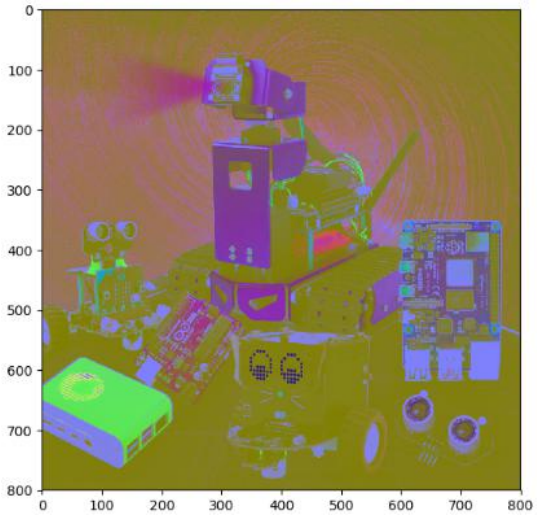
```
# cv2.waitKey(0)



img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)

#plt Display effect of two pictures before and after drawing

plt.figure(figsize=(14, 9), dpi=100)#Sets the size and pixels of the drawing area

plt.subplot(121)    # One row, two columns, the first

plt.imshow(img)

plt.subplot(122)    # One row, two columns, the second

#Color histogram equalization

plt.imshow(dst)

plt.show()
```

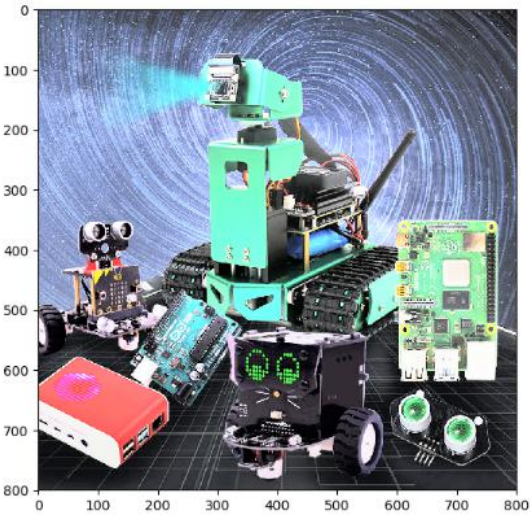





```
# YUV Histogram equalization

import cv2

import numpy as np
```

```
img = cv2.imread('yahboom.jpg',1)

imgYUV = cv2.cvtColor(img,cv2.COLOR_BGR2YCrCb)

# cv2.imshow('src',img)

channelYUV = cv2.split(imgYUV)

channelYUV[0] = cv2.equalizeHist(channelYUV[0])

channels = cv2.merge(channelYUV)

result = cv2.cvtColor(channels,cv2.COLOR_YCrCb2BGR)

# cv2.imshow('dst',result)

# cv2.waitKey(0)



imgYUV = cv2.cvtColor(imgYUV, cv2.COLOR_BGR2RGB)

result = cv2.cvtColor(result, cv2.COLOR_BGR2RGB)

#plt Display effect of two pictures before and after drawing

plt.figure(figsize=(14, 9), dpi=100)#Sets the size and pixels of the drawing area

plt.subplot(121)    # One row, two columns, the first

plt.imshow(imgYUV)

plt.subplot(122)    # One row, two columns, the second

#Color histogram equalization

plt.imshow(result)

plt.show()
```





3. Picture patching

Image restoration is a kind of algorithm in computer vision, whose goal is to fill the area in the image or video. This area is identified using a binary mask, and filling is usually completed according to the boundary information of the area to be filled. The most common application of image restoration is to restore old scanned photos. It is also used to delete small unwanted objects in the image.

dst = cv2.inpaint(src, inpaintMask, inpaintRadius, flags)

Parameter definition:

src：Source image

inpaintMask：A binary mask indicating the pixels to be repaired.

dst：Result image

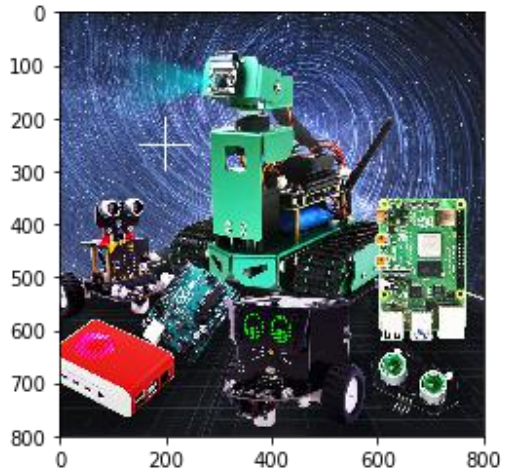
inpaintRadius：Indicates the radius of the repair

flags : Repair algorithm， Mainly:INPAINT_NS (Navier-Stokes based method) or INPAINT_TELEA (Fast marching based method)

Navier Stokes based fixes should be slower and tend to produce more ambiguous results than the fast marching method. In practice, we have not found this situation. INPAINT_ NS produced better results in our test and was also slightly faster than inpaint_ TELEA。

```
#Try to add a break to the original

import cv2

import numpy as np

import matplotlib.pyplot as plt



img = cv2.imread('yahboom.jpg',1)
```

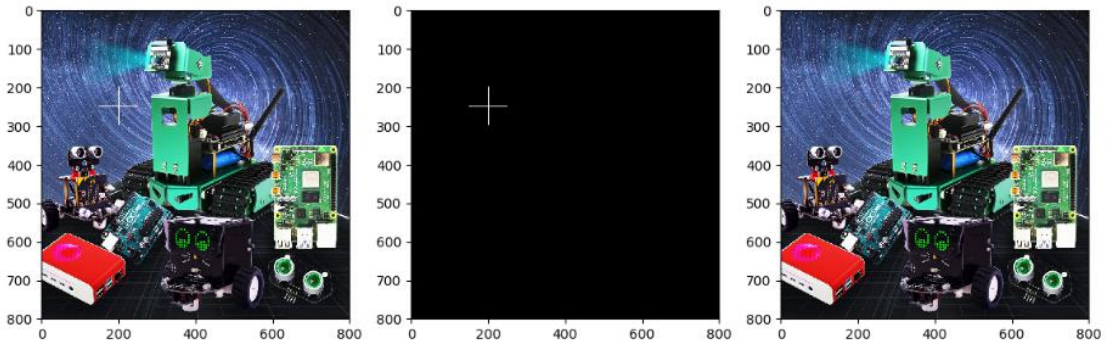
```
for i in range(200,300):
        img[i,200] = (255,255,255)
        img[i,200+1] = (255,255,255)
        img[i,200-1] = (255,255,255)
for i in range(150,250):
        img[250,i] = (255,255,255)
        img[250+1,i] = (255,255,255)
        img[250-1,i] = (255,255,255)
cv2.imwrite('damaged.jpg',img)
# cv2.imshow('image',img)
# cv2.waitKey(0)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img)
        plt.show()
```



#1 Broken diagram 2 Mask 3 inpaint

```
import cv2

import numpy as np

import matplotlib.pyplot as plt



img = cv2.imread('damaged.jpg',1)

#cv2.imshow('src',img)

imgInfo = img.shape

height = imgInfo[0]

width = imgInfo[1]

paint = np.zeros((height,width,1),np.uint8)



for i in range(200,300):

        paint[i,200] = 255

        paint[i,200+1] = 255

        paint[i,200-1] = 255

for i in range(150,250):

        paint[250,i] = 255

        paint[250+1,i] = 255

        paint[250-1,i] = 255

#cv2.imshow('paint',paint)

#1 src 2 mask
```

```
imgDst = cv2.inpaint(img,paint,3,cv2.INPAINT_TELEA)



# cv2.imshow('image',imgDst)

# cv2.waitKey(0)

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

paint = cv2.cvtColor(paint, cv2.COLOR_BGR2RGB)

imgDst = cv2.cvtColor(imgDst, cv2.COLOR_BGR2RGB)




plt.figure(figsize=(14, 9), dpi=100)#Sets the size and pixels of the drawing area

plt.subplot(131)    # The first of three columns in a row

plt.imshow(img)

plt.subplot(132)    # The second of three columns in a row

plt.imshow(paint)

plt.subplot(133)    # The third of three columns in a row

plt.imshow(imgDst)



      plt.show()
```



4. Brightness enhancement

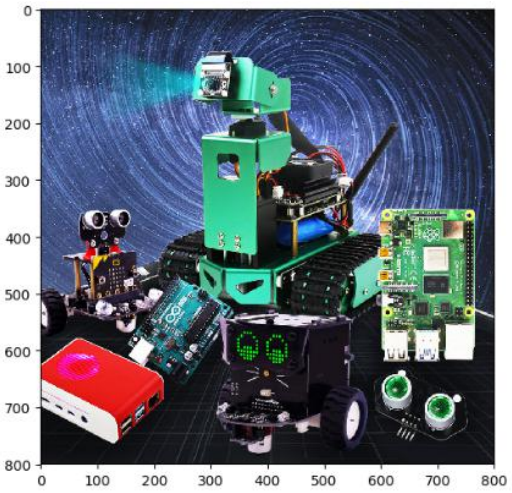
Implementation process: synchronously amplify the three channel values of each pixel while keeping the channel value between 0-255

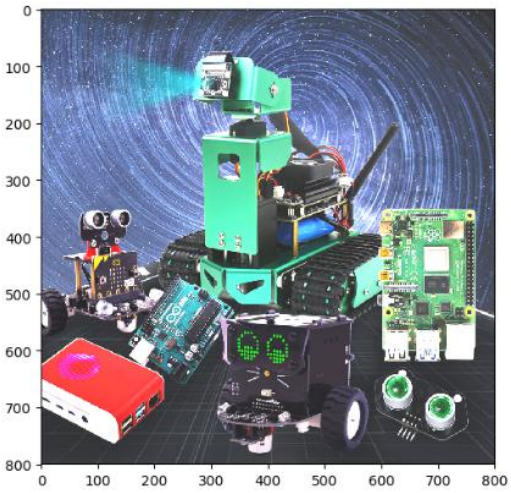
Map (F, list), apply the function f to the entire list, and return a new list

np. Clip (a, a_min, a_max, out = none), limiting the elements in a to the minimum value and the maximum value, and assigning values beyond this interval to the minimum value or the maximum value.

```
import cv2

import numpy as np

import matplotlib.pyplot as plt



img = cv2.imread('yahboom.jpg',1)

imgInfo = img.shape

height = imgInfo[0]

width = imgInfo[1]

#cv2.imshow('src',img)

dst = np.zeros((height, width, 3),np.uint8)

for i in range(0, height):

    for j in range(0, width):
```

```
            (b,g,r) = img[i,j]
            bb = int(b) + 40
            gg = int(g) + 40
            rr = int(r) + 40
            if bb>255:
                bb = 255
            if gg>255:
                gg = 255
            if rr>255:
                rr = 255
            dst[i,j] = (bb,gg,rr)
# cv2.imshow('dst',dst)
# cv2.waitKey(0)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(14, 6), dpi=100) #Sets the size and pixels of the drawing area
plt.subplot(121)    # One row, two columns, the first
plt.imshow(img)
plt.subplot(122)    # One row, two columns, the second
plt.imshow(dst)
plt.show()
```



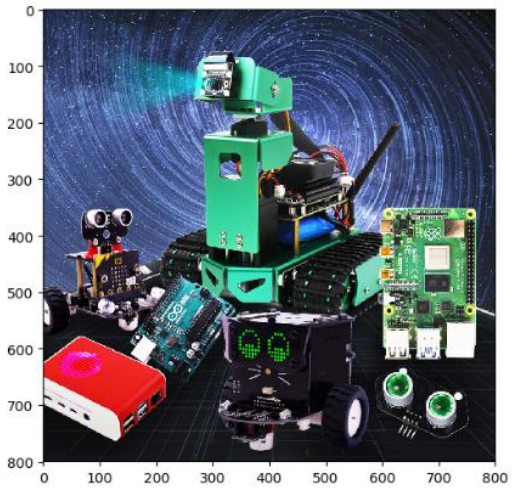


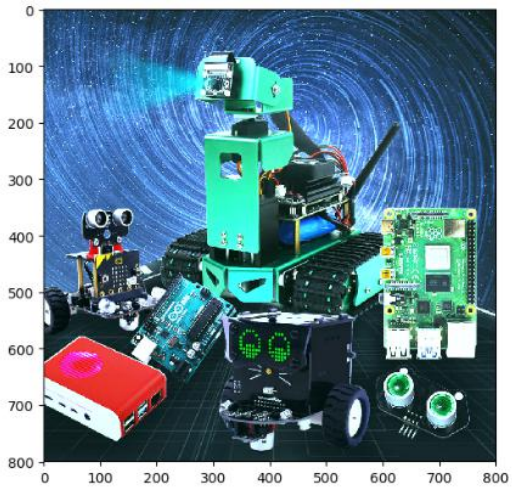
5. Skin whitening

Whitening formula for pictures： p = P*1.4(a)+ b;

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


img = cv2.imread('image0.jpg',1)

imgInfo = img.shape

height = imgInfo[0]

width = imgInfo[1]

#cv2.imshow('src',img)

dst = np.zeros((height,width,3),np.uint8)

for i in range(0,height):

    for j in range(0,width):

        (b,g,r) = img[i,j]
```

```
            bb = int(b*1.3) + 10
            gg = int(g*1.2) + 15

            if bb>255:
                bb = 255
            if gg>255:
                gg = 255

            dst[i,j] = (bb,gg,r)
# cv2.imshow('dst',dst)
# cv2.waitKey(0)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(14, 6), dpi=100) #Sets the size and pixels of the drawing area
plt.subplot(121)    # One row, two columns, the first
plt.imshow(img)
plt.subplot(122)    # One row, two columns, the second
plt.imshow(dst)
plt.show()
```



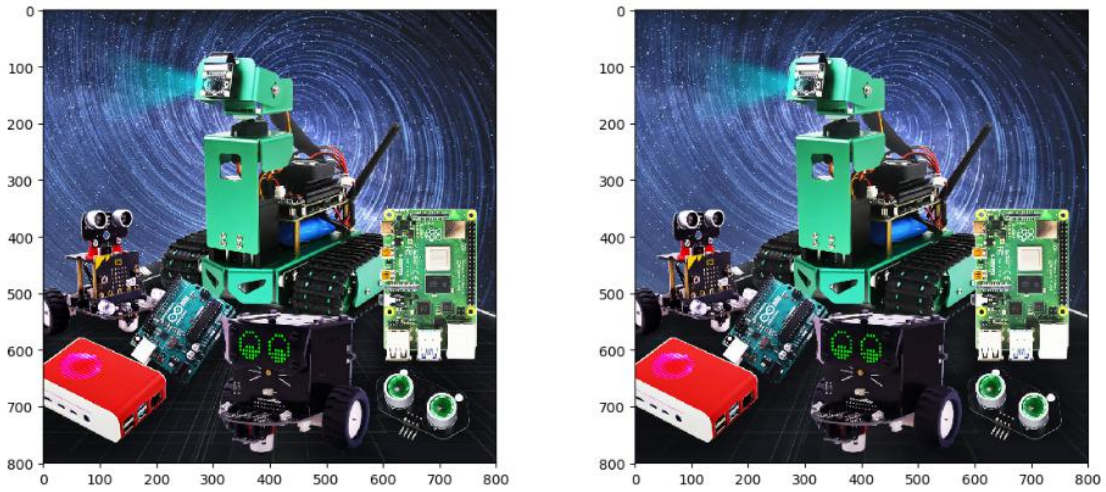


Bilateral filtering is a non-linear filtering method. It is a compromise processing combining the spatial proximity and pixel value similarity of an image. At the same time, it takes into account the spatial and information and gray similarity to achieve the purpose of edge preserving and denoising. It has the characteristics of simple, non iterative and local processing. The reason why the filtering effect of edge preserving denoising can be achieved is that the filter is composed of two functions: one is that the filter coefficient is determined by the geometric space distance, and the other is that the filter coefficient is determined by the pixel difference.

Generally speaking, the bilateral filter template is mainly generated by two templates. The first is a Gaussian template, and the second is a template generated by taking the difference of gray level as the function coefficient. Then, the two templates are dot multiplied to obtain the final bilateral filter template. The first template is a global template, so it only needs to be generated to the West. The second template needs to be calculated once for each pixel. The bilateral filter has a Gaussian variance sigma-d more than the Gaussian filter, which is a Gaussian filter function based on spatial distribution. Therefore, near the edge, the pixels far away from the edge will not affect the pixels on the edge too much, so the pixel values near the edge can be saved. However, due to the storage of too much high-frequency information, the bilateral filter cannot filter out the high-frequency noise in the color image, Only low-frequency information can be better filtered.

```
import cv2

import matplotlib.pyplot as plt



img = cv2.imread('yahboom.jpg',1)

#cv2.imshow('src',img)

dst = cv2.bilateralFilter(img,15,35,35)

# cv2.imshow('dst',dst)
```

```
# cv2.waitKey(0)

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(14, 6), dpi=100) #Sets the size and pixels of the drawing area

plt.subplot(121)    # One row, two columns, the first

plt.imshow(img)

plt.subplot(122)    # One row, two columns, the second

plt.imshow(dst)

plt.show()
```



6. Image processing - filtering

Filtering: it is a basic task in signal and image processing. Its purpose is to selectively extract some important information from the image according to different application environments. Filtering can remove noise in an image, extract visual features of interest, allow image resampling, and so on.

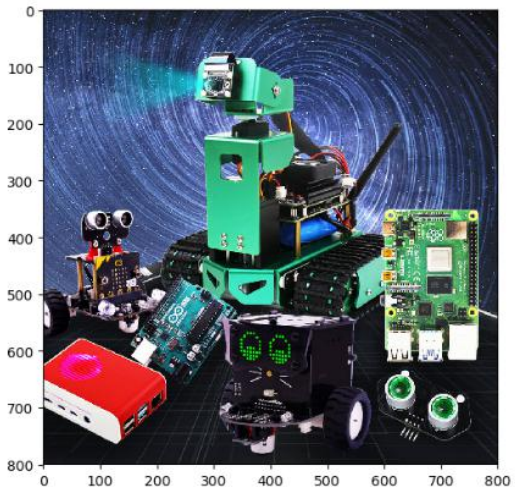
Frequency domain analysis: divide the image into different parts from low frequency to high frequency. The low frequency corresponds to the area where the image intensity changes little, while the high frequency corresponds to the area where the image intensity changes very much.

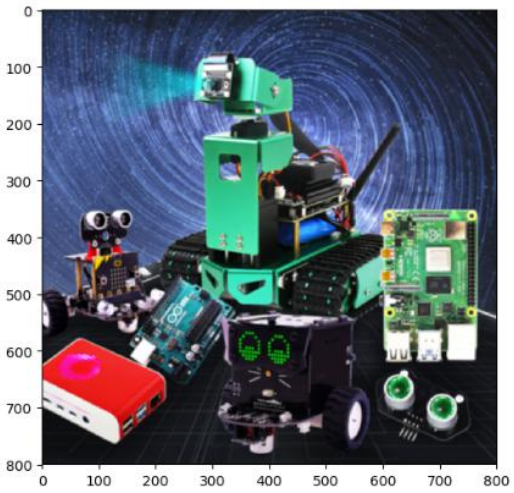
In the framework of frequency analysis, a filter is an operation used to enhance a certain band or frequency in an image and block (or reduce) other frequency bands. The low-pass filter eliminates the high-frequency part of the image, but retains the low-frequency part. The high pass filter eliminates the low frequency part.

Filtering (high pass, low pass, band pass, band stop), blurring, denoising, smoothing, etc.

Gaussian blur is essentially a mean blur, but Gaussian blur is weighted average. The closer the distance, the greater the weight of the point, and the farther the distance, the smaller the weight of the point.
Generally speaking, Gaussian filtering is the process of weighted average of the whole image. The value of each pixel is obtained by weighted average of itself and other pixel values in the neighborhood.

```
import cv2

import numpy as np

import matplotlib.pyplot as plt



img = cv2.imread('yahboom.jpg',1)

# cv2.imshow('src',img)

dst = cv2.GaussianBlur(img,(5,5),1.5)

# cv2.imshow('dst',dst)

# cv2.waitKey(0)

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(14, 6), dpi=100) #Sets the size and pixels of the drawing area

plt.subplot(121)    # One row, two columns, the first

plt.imshow(img)

plt.subplot(122)    # One row, two columns, the second

plt.imshow(dst)

plt.show()
```





7. median filtering

Median filtering: image smoothing can eliminate salt and pepper noise. The basic idea is to traverse the image through the filter and take the median value of the pixel value of each filter area as the new pixel value.

The algorithm idea is as follows:

(1) Input image and turn gray;

(2) Add salt and pepper noise to the grayscale image

(3) Traversing the pixel points and putting the pixel values in the filter area into a one-dimensional array;

(4) The one-dimensional array is selected and sorted, and the intermediate value is assigned to the filter center, that is, the pixel points of the original image traversed are changed to the median value of the filter area;

(5) And outputs the median filtered image.

```
# Median filter (3 * 3)

import cv2

import numpy as np

import matplotlib.pyplot as plt



img = cv2.imread('yahboom.jpg',1)

imgInfo = img.shape

height = imgInfo[0]
```

```
width = imgInfo[1]

img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)

# cv2.imshow('src',img)

dst = np.zeros((height,width,3),np.uint8)

collect = np.zeros(9,np.uint8)

for i in range(1,height-1):

    for j in range(1,width-1):

        k = 0

        for m in range(-1,2):

            for n in range(-1,2):

                gray = img[i+m,j+n]

                collect[k] = gray

                k = k+1

        # 0 1 2 3 4 5 6 7 8

        #     1

        for k in range(0,9):

            p1 = collect[k]

            for t in range(k+1,9):

                if p1<collect[t]:

                    mid = collect[t]

                    collect[t] = p1
```

```
                    p1 = mid

            dst[i,j] = collect[4]

# cv2.imshow('dst',dst)

# cv2.waitKey(0)

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(14, 6), dpi=100) #Sets the size and pixels of the drawing area

plt.subplot(121)    # One row, two columns, the first

plt.imshow(img)

plt.subplot(122)    # One row, two columns, the second

plt.imshow(dst)

plt.show()
```