Geometric transformation

1.Picture zoom

In opencv, the function of image scaling is:cv2.resize(InputArray src, OutputArray dst, Size, fx, fy, interpolation)

Parameter explanation:

InputArray src	Input picture
OutputArray dst	Output picture
Size	Output picture size
fx, fy	Scaling factor along x-axis and y-axis
interpolation	Insert by:

interpolation Option to use：

INTER_NEAREST	Nearest neighbor interpolation
INTER_LINEAR	Bilinear interpolation (default)
INTER_AREA	Resampling is performed using the pixel region relationship.
INTER_CUBIC	Bicubic interpolation of 4x4 pixel neighborhood
INTER_LANCZOS4	Lanczos interpolation in the neighborhood of 8x8 pixels

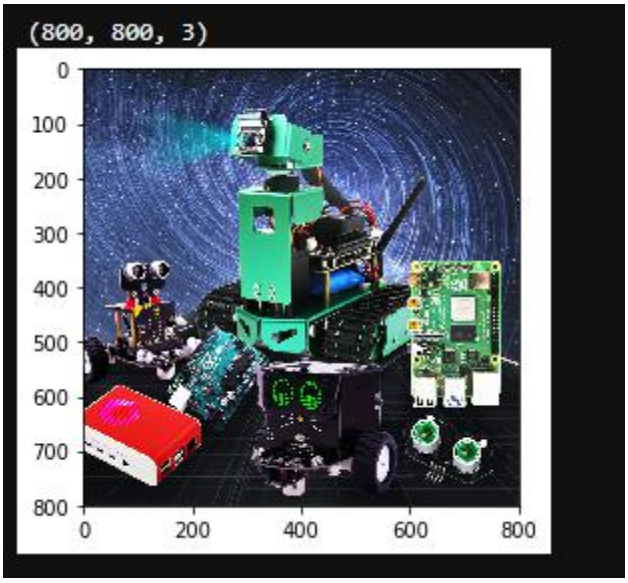
Be careful:

1. Output size format is (width, height)
2. The default interpolation method is bilinear interpolation

```
# 1 load 2 info 3 resize 4 check
import cv2
import matplotlib.pyplot as plt    #Python's 2D drawing library
# Read in original picture
img = cv2.imread('yahboom.jpg')

# Print out picture size
print(img.shape)

# Assign the height and width of the picture to X and Y respectively
x, y = img.shape[0:2]

# Display original image
#cv.imshow('OriginalPicture', img)

# Zoom to the original half, and the output size format is (width, height)
img_test1 = cv2.resize(img, (int(y / 2), int(x / 2)))
```

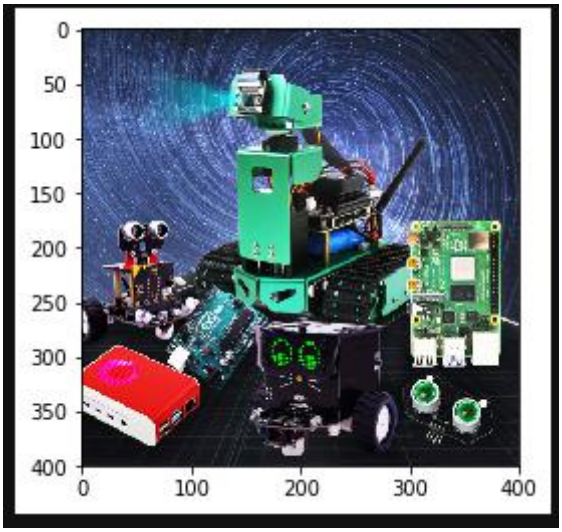
```
# cv2.imshow('resize0', img_test1)
# cv2.waitKey()

# Nearest neighbor interpolation scaling
# Zoom to the original quarter
img_test2 = cv2.resize(img, (0, 0), fx=0.25, fy=0.25, interpolation=cv2.INTER_NEAREST)
# cv.imshow('resize1', img_test2)
# cv.waitKey()
# cv.destroyAllWindows()
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
dst1 = cv2.cvtColor(img_test1, cv2.COLOR_BGR2RGB)
dst2 = cv2.cvtColor(img_test2, cv2.COLOR_BGR2RGB)

# Show original image
plt.imshow(img)
plt.show()
```

After execution, you can see that the image is 800 * 800



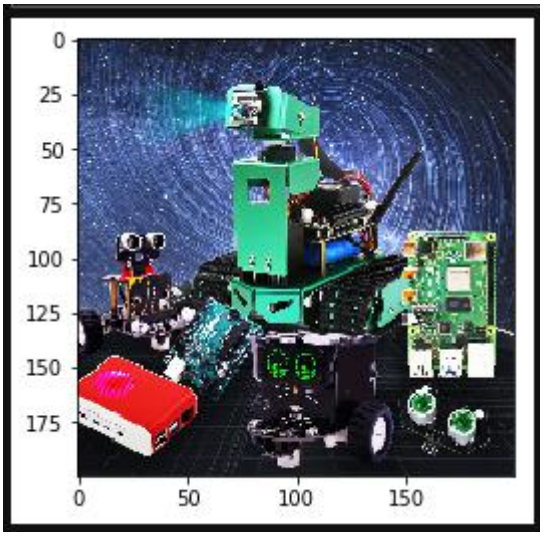

```
# Display zoom 1 / 2
plt.imshow(dst1)
plt.show()
```

After execution, you can see that the image is 400 * 400, scaled by half.




```
# Display scaling 1 / 4 adjacent interpolation scaling
plt.imshow(dst2)
plt.show()
```

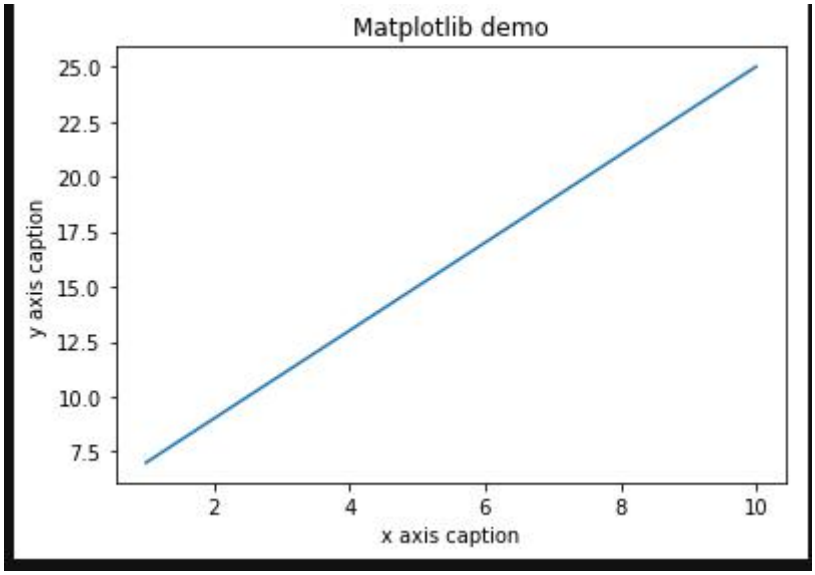
After execution, you can see that the image is 200 * 200, scaled by one quarter.



Let's talk about Matplotlib: Python's 2D drawing library.

Reference tutorial: https://www.runoob.com/numpy/numpy-matplotlib.html

```
import numpy as np
from matplotlib import pyplot as plt
x = np.arange(1,11)
y =  2 * x +  5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x,y)
plt.show()
```



3.Picture clipping

Image clipping first reads the image, and then obtains the pixel region in the array. In the following code, select the shape area X: 300-500 Y: 500-700. Note that the image size is 800 * 800, so the selected area should not exceed this resolution.

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

dst = img[500:700,300:500]    #Here, select the rectangular area: X: 300-500, Y: 500-700

#cv2.imshow('image',dst)
#cv2.waitKey(0)
```

The following will display two kinds of compressed images in jupyterlab control:

```
#Bgr8 to JPEG format
import enum
import cv2

def bgr8_to_jpeg(value, quality=75):
    return bytes(cv2.imencode('.jpg', value)[1])
```

The following images are compared before and after:

```
import ipywidgets.widgets as widgets

image_widget1 = widgets.Image(format='jpg', )
image_widget2 = widgets.Image(format='jpg', )

# display the container in this cell's output
display(image_widget1)
```

```
display(image_widget2)
img1 = cv2.imread('yahboom.jpg',1)
image_widget1.value = bgr8_to_jpeg(img1)    #original image
image_widget2.value = bgr8_to_jpeg(dst)      #Clipped image
```

3.Picture pan

The conversion of the original image SRC into the target image DST is realized by the conversion matrix M:

dst(x, y) = src(M11x + M12y+M13, M21x+M22y+M23)

If the original image SRC is moved 200 pixels to the right and 100 pixels to the right, the corresponding relationship is:

dst(x, y) = src(x+200, y+100)

Complete the above expression, namely:

dst(x, y) = src(1·x + 0·y + 200, 0·x + 1·y + 100)

According to the above expression, it can be determined that the values of each element in the corresponding transformation matrix M are:

M11=1

M12=0

M13=200
M21=0
M22=1
M23=100
Substituting the above values into the transformation matrix M, we obtain:

M = []

Next, directly use the transformation matrix m to call the function CV2. Warpaffine() to complete the translation of the image.

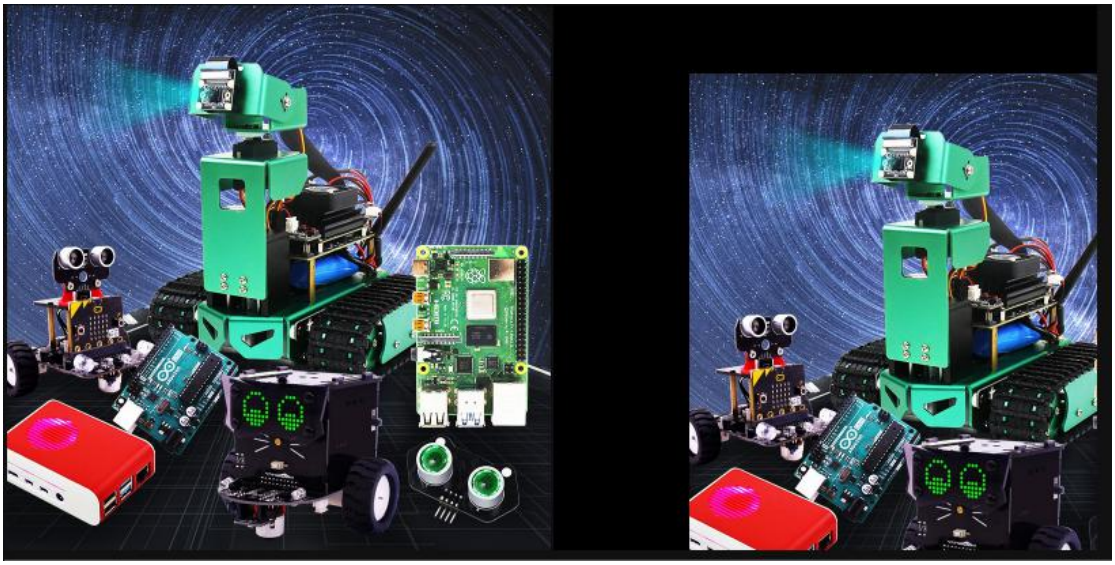
```
import cv2
import numpy as np
img = cv2.imread('yahboom.jpg',1)
#cv2.imshow('src',img)
imgInfo = img.shape
height = imgInfo[0]
width = imgInfo[1]
####
matShift = np.float32([[1,0,200],[0,1,100]])# 2*3
dst = cv2.warpAffine(img, matShift, (height, width))        #1 data 2 mat 3 info
# Shift matrix
# cv2.imshow('dst',dst)
# cv2.waitKey(0)
```

The following will display the comparison between the original image and the translated image in the jupyterlab control:

```
#Bgr8 to JPEG format
import enum
import cv2

def bgr8_to_jpeg(value, quality=75):
    return bytes(cv2.imencode('.jpg', value)[1])
```

```
import ipywidgets.widgets as widgets

image_widget1 = widgets.Image(format='jpg', )
image_widget2 = widgets.Image(format='jpg', )
# create a horizontal box container to place the image widget next to eachother
image_container = widgets.HBox([image_widget1, image_widget2])

# display the container in this cell's output
display(image_container)
#display(image_widget2)

img1 = cv2.imread('yahboom.jpg',1)
```

```
image_widget1.value = bgr8_to_jpeg(img1)
image_widget2.value = bgr8_to_jpeg(dst)
```



It can be seen from the image that the picture has moved to the lower right corner (200,100).

4.Image mirroring

There are two kinds of image mirroring: horizontal mirroring and vertical mirroring. The horizontal mirror image takes the vertical center line of the image as the axis to exchange the pixels of the image, that is, to exchange the left half and the right half of the image. The vertical mirror image is to take the horizontal center line of the image as the axis to exchange the upper half of the image and the off-duty part.

Transformation principle:

Let the width of the image be width and the length be height. (x, y) is the coordinate after transformation, and (x0, Y0) is the coordinate of the original image

Horizontal mirror transformation

$$x = width - x_0 - 1$$
$$y = y_0$$

Forward mapping

The inverse transformation is:

$$x_0 = width - x - 1$$
$$y_0 = y$$

Backward mapping

Vertical mirror transform

$$x = x_0$$
$$y = height - y_0 - 1$$

The inverse transformation is:

$$x_0 = x$$
$$y_0 = height - y - 1$$

Summary:

In the horizontal mirror transformation, the whole image is traversed, and then each pixel is processed according to the mapping relationship. In fact, horizontal mirror transformation is to change the column of image coordinates to the right and the column on the right to the left, which can be transformed in columns. The same is true for vertical mirror transformation, which can be transformed in units of behavior.

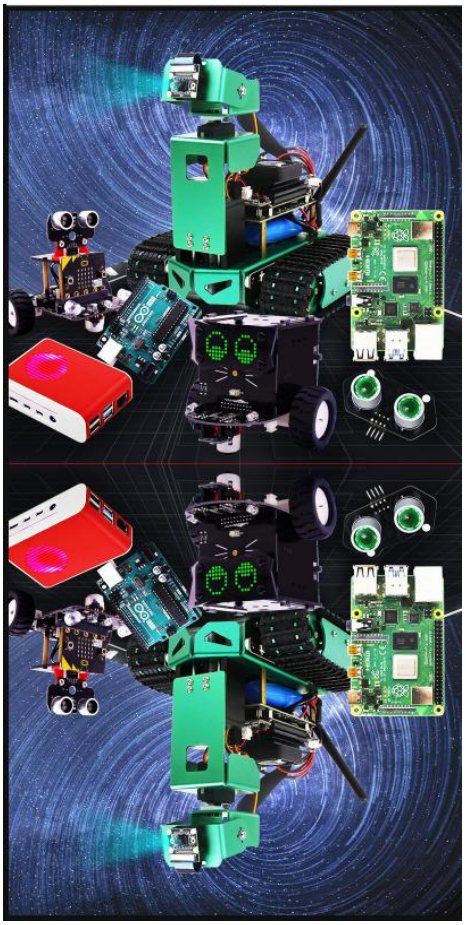
Here we take vertical transformation as an example to see how Python writes:

```
import cv2
import numpy as np
img = cv2.imread('yahboom.jpg',1)
#cv2.imshow('src',img)
imgInfo = img.shape
height = imgInfo[0]
width = imgInfo[1]
deep = imgInfo[2]
newImgInfo = (height*2,width,deep)
dst = np.zeros(newImgInfo,np.uint8)#uint8
for i in range(0,height):
    for j in range(0,width):
        dst[i,j] = img[i,j]
        #x y = 2*h - y -1
        dst[height*2-i-1,j] = img[i,j]
for i in range(0,width):
    dst[height,i] = (0,0,255) #BGR
```

```
#Bgr8 to JPEG format
import enum
import cv2

def bgr8_to_jpeg(value, quality=75):
    return bytes(cv2.imencode('.jpg', value)[1])
```

```
import ipywidgets.widgets as widgets

image_widget1 = widgets.Image(format='jpg', )
# image_widget2 = widgets.Image(format='jpg', )
# create a horizontal box container to place the image widget next to eachother
# image_container = widgets.HBox([image_widget1, image_widget2])
```

```
# display the container in this cell's output
display(image_widget1)
#display(image_widget2)
image_widget1.value = bgr8_to_jpeg(dst)
```



5. affine transformation

Affine transformation or affine map is a linear transformation from two-dimensional coordinates (x, y) to two-dimensional coordinates (U, V). Its mathematical expression is as follows:

$$\begin{cases} u = a_1 x + b_1 y + c_1 \\ v = a_2 x + b_2 y + c_2 \end{cases}$$

The corresponding homogeneous coordinate matrix is expressed as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine transformation maintains the "straightness" (straight lines are still straight lines after affine transformation) and "parallelism" (the relative positional relationship between straight lines remains unchanged, parallel lines are still parallel lines after affine transformation, and the position order of points on the straight lines will not change). Three pairs of non collinear corresponding points determine a unique affine transformation.

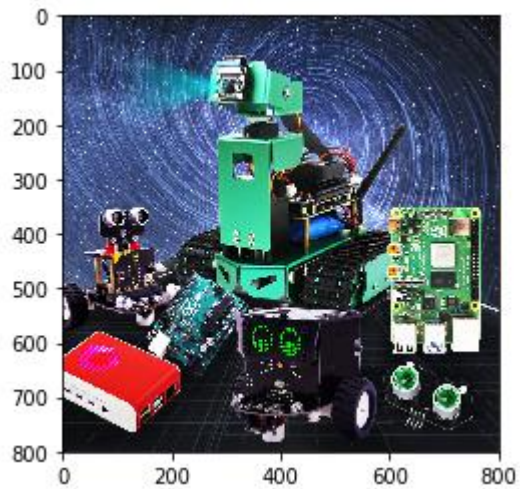
The rotation and lifting of the image is the affine transformation of the image. The affine transformation also requires an M matrix. However, because the affine transformation is complex, it

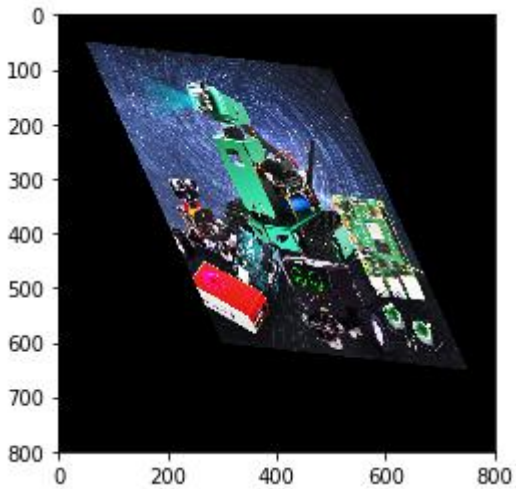
is difficult to find this matrix directly. Opencv provides the automatic solution of m according to the correspondence between the three points before and after the transformation. This function is M = CV2. Getaffinetransform (pos1, pos2). The two positions are the corresponding positions before and after the transformation. The output is the affine matrix M. Then use the function CV2. Warpaffine().

Let's take a look at the use block diagram of affine transformation and perspective transformation: two methods of image transformation: cv2.warpaffine and cv2.warpsperspective

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

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

img_bgr2rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img_bgr2rgb)
plt.show()
# cv2.waitKey(0)
```

```
imgInfo = img.shape
height = imgInfo[0]
width = imgInfo[1]
#src 3->dst 3 (upper left corner lower left corner upper right corner)
matSrc = np.float32([[0,0], [0,height-1], [width-1,0]])
matDst = np.float32([[50,50], [300,height-200], [width-300,100]])
#Combine
matAffine = cv2.getAffineTransform(matSrc,matDst)# mat 1 src 2 dst
dst = cv2.warpAffine(img,matAffine,(width,height))
img_bgr2rgb = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
plt.imshow(img_bgr2rgb)
```

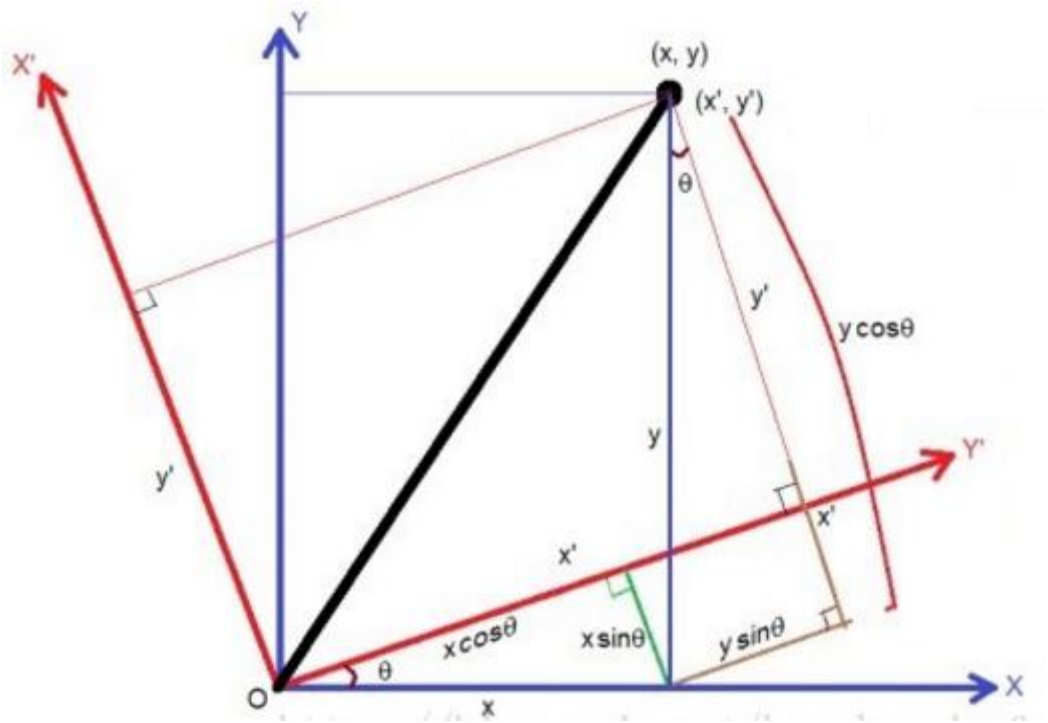




6. pictures rotating

Image rotation refers to the process in which an image is rotated by a certain angle according to a certain position, and the original size of the image is still maintained during the rotation. After the image is rotated, the horizontal symmetry axis, the vertical symmetry axis and the center coordinate origin of the image may be transformed, so the coordinates in the image rotation need to be transformed accordingly.

As shown below:



Suppose the image rotates counterclockwise θ， The rotation conversion can be obtained according to the coordinate conversion as follows:

$$\begin{cases} x' = r\cos(\alpha - \theta) \\ y' = r\sin(\alpha - \theta) \end{cases} \quad (1)$$

and

$$r = \sqrt{x^2 + y^2}, \sin\alpha = \frac{y}{\sqrt{x^2 + y^2}}, \cos\alpha = \frac{x}{\sqrt{x^2 + y^2}} \quad (2)$$

（2）Bring in (1) to get:

$$\begin{cases} x' = x\cos\theta + y\sin\theta \\ y' = -x\sin\theta + y\cos\theta \end{cases} \quad (3)$$

Namely as follows:

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And the gray value of the rotated image is equal to the gray value of the corresponding position in the original image as follows:

f(x',y')=f(x,y)

The above is the principle of rotation, but the API provided by opencv can directly obtain the transformation matrix through functions. The syntax format of this function is:

matRotate = cv2.getRotationMatrix2D(center, angle, scale)

center：Center point of rotation

angle： The angle of rotation. Positive numbers are counter clockwise; Negative numbers are clockwise.

scale： Transform scale (scale size). 1 is not changed, less than 1 is reduced, and greater than 1 is enlarged.

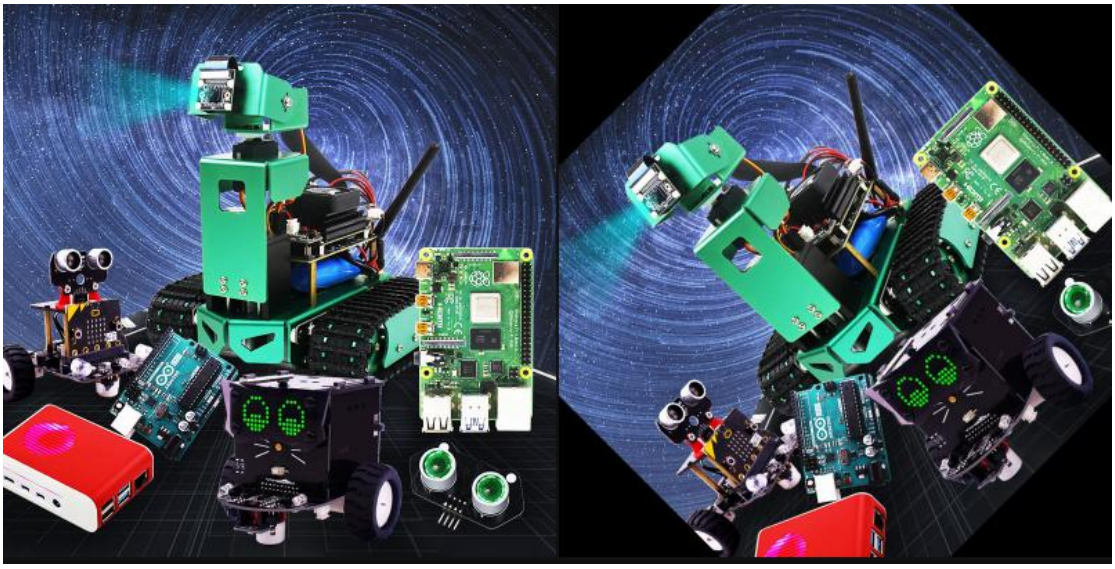
```
import cv2
import numpy as np
img = cv2.imread(yahboom.jpg',1)
#cv2.imshow('src',img)
imgInfo = img.shape
height = imgInfo[0]
width = imgInfo[1]
matRotate = cv2.getRotationMatrix2D((height*0.5, width*0.5), 45, 1)# mat rotate 1 center 2 angle 3 scale
#100*100 25
dst = cv2.warpAffine(img, matRotate, (height,width))
```

In the following, the original image and the rotated image will be displayed in the jupyterlab control

```
#bgr8 to jpeg format
import enum
import cv2

def bgr8_to_jpeg(value, quality=75):
    return bytes(cv2.imencode('.jpg', value)[1])
```

```
import ipywidgets.widgets as widgets

image_widget1 = widgets.Image(format='jpg', )
image_widget2 = widgets.Image(format='jpg', )
# create a horizontal box container to place the image widget next to eachother
image_container = widgets.HBox([image_widget1, image_widget2])

# display the container in this cell's output
```

```
display(image_container)
#display(image_widget2)

img1 = cv2.imread('image0.jpg',1)

image_widget1.value = bgr8_to_jpeg(img1)
      image_widget2.value = bgr8_to_jpeg(dst)
```



7. Perspective transformation

Perspective transformation is also called projection transformation. The affine transformation we often say is a special case of perspective transformation. The purpose of perspective transformation is to transform objects that are straight lines in reality, which may appear as oblique lines in the picture, into straight lines through perspective transformation. Perspective transformation can map the rectangle to any quadrilateral. We will use this technology later when the robot drives automatically.

Perspective transformation through function:

dst = cv2. warpPerspective(src, M, dsize[,flag, [,borderMode[,borderValue]]])

dst : Output image after perspective transformation，dsize:Determines the actual size of the output。

src：Source image

M：Transformation matrix of 3x3

dsize: Output image size.

flags：Interpolation method, default inter_ Linear (bilinear interpolation), when it is warp_ INVERSE_ Map, which means that M is an inverse transform and can realize the inverse transform from the target DST to Src.

borderMode： Edge type. The default is border_ CONSTANT. When the value is border_ When transmitting, the values in the target image do not change, and these values correspond to the abnormal values in the original image.

borderValue：Boundary value, default to 0

Like affine transformation, opencv still provides a function CV2. Getperspectivetransform() to provide the transformation matrix for the above.

The function is as follows:

matAffine = cv2.getPerspectiveTransform(matSrc, matDst)

matSrc：Enter the four vertex coordinates of the image.

matDst：Four vertex coordinates of the output image.

```
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]
#src 4->dst 4(upper left corner  lower left corner  upper right corner  lower right corner)
matSrc = np.float32([[200,100],[200,400],[600,100],[width-1,height-1]])
matDst = np.float32([[200,200],[200,300],[500,200],[500,400]])
#combination
matAffine = cv2.getPerspectiveTransform(matSrc,matDst)# mat 1 src 2 dst
dst = cv2.warpPerspective(img,matAffine,(width,height))
img_bgr2rgb = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
plt.imshow(img_bgr2rgb)
    plt.show()
```