Lesson 4 Image processing methods Introduction

1. Image preprocessing

Usually, we need to perform special processing on the original image. For example, when doing facial recognition, we often need to convert the image to a grayscale image. Before formally learning image processing, let's first look at the conversion of different color spaces in OpenCV.

There are hundreds of methods for converting between different color spaces in OpenCV. At present, there are three common color spaces in the field of computer vision: grayscale, BGR, HSV (tone Hue, saturation, brightness value). The YCrCba color space is useful in our gameplay course, where Y refers to the brightness, Cr refers to the difference between the red part of the RGB input signal and the brightness value of the RGB signal, and Cb refers to the blue part of the RGB input signal The difference between the brightness value and the RGB signal. The feature of this color space is to separate brightness and chroma, which is suitable for image processing and is often used in human skin color detection.

Color conversion: cv2.cvtColor(Img,Transform_model).

Img-image object.

Transform_model-Specifies the color conversion mode.

Note that the color and light mixing here is different from the pigment mixing. The pigment has color because it reflects the light of that color. When multiple colors are mixed, they follow the subtractive color mixing, and the light on the screen is the light emitted by the screen, which follows the principle of color enhancement.

2. Image Processing Principles

In OpenCV, most of the processing of images and videos involves the concept of Fourier transform. Fourier observed that all waveform in mathematics can be obtained by superimposing a series of simple sinusoids with different frequencies. It is inferred and proved that any waveform that people see is obtained by superposition of other waveform.

When we process images, we can get the area of interest by removing part of the waveform.

In addition to Fourier transform, high-pass filters and low-pass filters are often mentioned in the image processing process.

Among them, the high-pass filter is a filter that detects a certain area of the image, and then enhances the brightness of the pixel according to the brightness difference between the pixel and the surrounding pixels.

The high-pass filter will increase the brightness gap between pixels and surrounding pixels, and is generally used in edge detection.

Low-pass filtering is a filter that smoothes the brightness of a pixel when the brightness difference between a pixel and surrounding pixels is less than a certain value. It is mainly used for denoising and blurring, such as the most commonly used blur filter (smoothing filter): Gaussian blur, which is essentially a low-pass filter that weakens the strength of high-frequency signals.

3. Edge detection

Each picture contains a lot of content, and it is often need to separate different content when processing images.

There is always an edge between two adjacent areas with different gray values, and the edge is the performance of the gray value discontinuity. In general, the purpose of edge detection is to detect the edge of an image, that is, where the image is significantly different, to prepare for the subsequent further image processing.

For example, if we want to find oranges from a picture that contains apples, oranges, and pears at the same time, then we need to first determine which of the objects are in the picture and which are the background. Here we are interested in fruits. Edge detection can detect the contour edges of fruits in the picture, helping the computer to determine which parts of the picture are objects and which parts are the background, and prepare for the subsequent further determination of which are oranges and which are not oranges.

OpenCV provides many edge detection filter functions, including Laplacian(), Sobel() and Scharr(). These filter functions will convert non-edge areas to black, and edge areas to white or other saturated colors.

However, in actual processing, these functions are easy to misidentify the noise as an edge, so it is necessary to blur the image before finding the edge to remove the noise. OpenCV also provides many blur filter functions, commonly used are blur() (simple arithmetic average), medianBlur() (median filtering), GaussianBlur().

There is also a very convenient Canny function in actual use, which can easily realize edge detection with only one line of code.

```
cv2.Canny(Pic,Minval,Maxval).
```

cv2.Canny(Pic,Minval,Maxval)。

Pic——The image object whose edges are to be detected.

Minval——The minimum gray gradient threshold. When the image gray gradient is lower than Minval, it will be discarded.

Maxval——The maximum gradient value threshold. When the image gray gradient is higher than Maxval, it will be considered as an edge. When the image gray gradient is between Minval and Maxval, if it is connected to the edge, it is part of the boundary, otherwise it is discarded.

The Canny edge detection algorithm actually uses the following 5 steps to process the image:

1) Use a Gaussian filter to smooth the image and filter out noise.

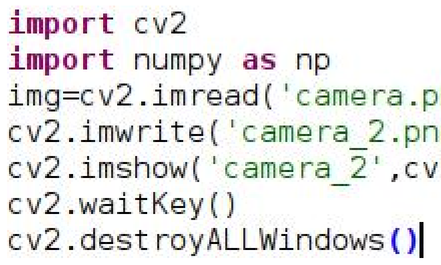
2) Calculate the gradient intensity and direction of each pixel in the image.

3) Use Non-Maximum Suppression on the edge to eliminate spurious response caused by edge detection.

4) Use dual thresholds on the detected edges to remove false positives, that is, determine which of the gradient values are between Minval and Maxval are edges and which should be discarded.

5) Analyze all edges and the connections between them, suppress isolated weak edges, to ensure that true edges are left and unobvious edges are eliminated.

Example: Create a new py file, and put a picture with the full name "camera.png" in the same folder of the py file, enter the following code, after running, you can see the white outline of the image appears on the screen, and other places change It is black, and the image is hidden after pressing any key.



```
import cv2
import numpy as np
img=cv2.imread('camera.png',0)
cv2.imwrite('camera_2.png',cv2.Canny(img,200,300))
cv2.imshow('camera_2',cv2.imread('camera_2.png'))
cv2.waitKey()
cv2.destroyALLWindows()
```

4. Contour detection

In computer vision, it is often necessary to detect the contours of objects in images or videos, and on this basis calculate polygon boundaries, shape approximations, and calculate regions of interest.

In OpenCV, you can call the findContours() function to detect contours. The specific parameters of the function are as follows:

findContours(Img,Model,Method)

Img——The picture object whose contour is to be detected. Note that the picture needs to be a grayscale picture.

Model——Indicates the detection mode of the contour:

cv2.RETR_EXTERNAL:It means that only the outer contour is detected.

cv2.RETR_LIST: The detected contour does not establish a hierarchical relationship.

cv2.RETR_CCOMP: Establish two levels of contours, the upper layer is the outer boundary, and the inner layer is the boundary information of the inner hole. If there is a connected object in the inner hole, the boundary of this object is also on the top layer.

cv2.RETR_TREE：Establish the outline of a hierarchical tree structure.

Method-Represents the contour approximation method and detection

strategy, specifically as follows:

cv2.CHAIN_APPROX_NONE stores all contour points, and the pixel position difference between two adjacent points does not exceed 1, that is, max(abs(x1-x2), abs(y2-y1))==1.

cv2.CHAIN_APPROX_SIMPLE compresses the elements in the horizontal, vertical, and diagonal directions, and only retains the end point coordinates in that direction. For example, a rectangular outline only needs 4 points to save the outline information.

cv2.CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS uses the teh-Chinl chain approximation algorithm.

The findContours() function has three return values: the modified image, the contour of the image, and their levels.

5. Line and circle detection

Hough transform is the theoretical basis behind straight line and shape detection. It was proposed by Richard Duda and Peter Hart in the 1960s.

Straight line detection can be done by either of HougLines and HoughLinesP. The first function uses the standard Hough transform, and the second function uses the probabilistic Hough transform.

The probabilistic Hough transform is an optimized version of the standard Hough transform. It only performs line detection by analyzing a subset of points and estimating the probability that these points belong to a straight line. This algorithm requires less calculation and is fast to execute.

Probability Hough transform:

cv2.HougLinesP(img,rho,theta,threshod,minLineLength,maxLineGa

p)

img——The image to be processed must be a grayscale image (binary image). Generally, the result image of canny edge detection is used.

rho——The distance accuracy of the line segment in pixels. For double type, 1.0 is recommended.

theta——The angular accuracy of the line segment in radians. numpy.pi/180 is recommended.

threshod——Threshold, all straight lines below this threshold will be deleted.

minLineLength——The minimum length of the line segment in pixels, set according to the application scenario.

maxLineGap——The maximum allowable gap (break) between two line segments in the same direction is judged to be a line segment. If it is less than the set value, the two line segments will be regarded as one line segment. The larger the value, the larger the allowable break on the line segment, the more likely it is Detect potential straight line segments.

OpenCV can use the HoughCircles function to detect circles, which is similar to using the HougLines function.

cv2.HoughCircles(image, method, dp, minDist, circles, param1, param2, minRadius, maxRadius)

image——input image, need grayscale image (binary image).

method——Detection method, commonly used CV_HOUGH_GRADIENT.

dp——in order to detect the reciprocal of the ratio of the resolution of the accumulator image at the inner circle center to the input image, such as dp=1, the accumulator and the input image have the same resolution, if dp=2, the accumulator has half of the input image That big width and height.

minDist-represents the minimum distance between the center of two circles.

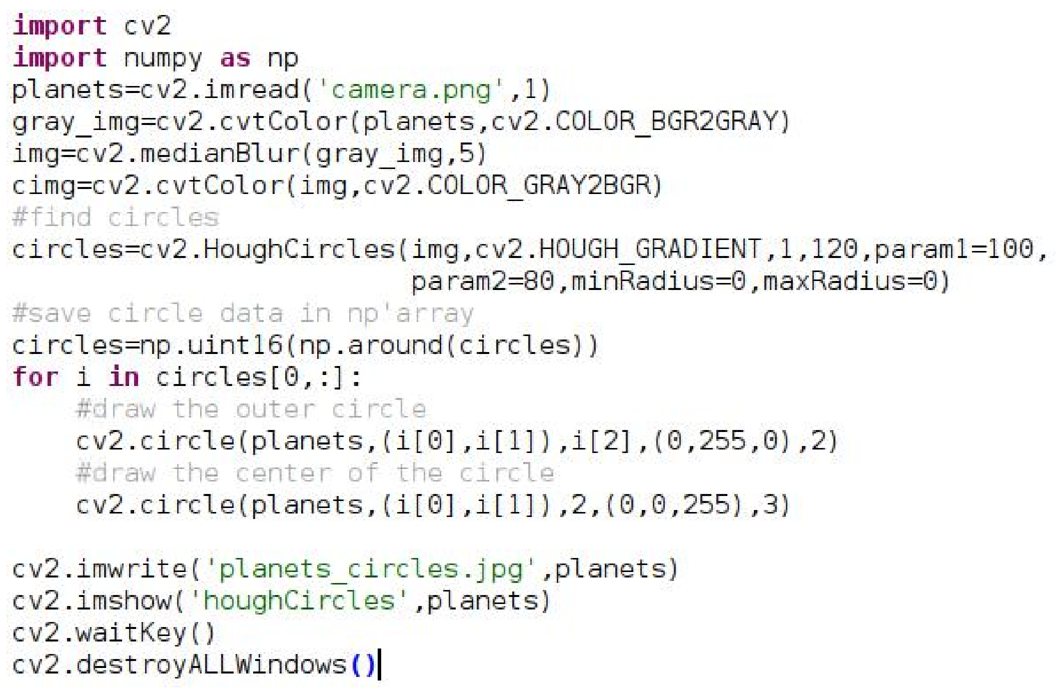
param1——The default value is 100, which is the corresponding parameter of the detection method set by the method. For the current only method, the Hough gradient method cv2.HOUGH_GRADIENT, it represents the high threshold passed to the canny edge detection operator, and the low threshold is high Half of the threshold

param2——The default value is 100. It is the corresponding parameter of the detection method set by the method. For the current only method Hough gradient method cv2.HOUGH_GRADIENT, it represents the accumulator threshold at the center of the detection stage. The smaller it is, the better More circles that don't exist at all are detected, and the larger the circle, the closer the circle can pass the detection to the perfect circle.

minRadius——The default value is 0, the minimum value of the circle radius.

maxRadius——The default value is 0, the maximum value of the circle radius.

Example: Create a new py file, and put a picture with the full name "camera.png" in the same folder of the py file, enter the following code, after running, you can see that the detected circle is marked with a green line on the screen Circle the outline and mark the center of the circle. Press any key to close the image window.



```python
import cv2
import numpy as np
planets=cv2.imread('camera.png',1)
gray_img=cv2.cvtColor(planets,cv2.COLOR_BGR2GRAY)
img=cv2.medianBlur(gray_img,5)
cimg=cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
#find circles
circles=cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,120,param1=100,
                         param2=80,minRadius=0,maxRadius=0)
#save circle data in np'array
circles=np.uint16(np.around(circles))
for i in circles[0,:]:
    #draw the outer circle
    cv2.circle(planets,(i[0],i[1]),i[2],(0,255,0),2)
    #draw the center of the circle
    cv2.circle(planets,(i[0],i[1]),2,(0,0,255),3)

cv2.imwrite('planets_circles.jpg',planets)
cv2.imshow('houghCircles',planets)
cv2.waitKey()
cv2.destroyALLWindows()
```

If you want to detect polygons, you can combine the cv2.findContours function and the cv2.approxPloyDP function.