**Practical Assignment №1**

**20321233 zou letian**

**20321308 cao xinyang**

**20321309 chen hanzheng**

1. Purpose

1. *Histograms.* Select an arbitrary low-contrast image. Perform

histogram alignment and contrast stretching, use the considered transformations and built-in MATLAB or OpenCV functions. Compare the results.

2. *Projections.* Select an arbitrary image containing monotone areas and prominent objects. Calculate image projections on the

vertical and horizontal axes. Define the objects boundaries.

3. *Profiles.* Select an arbitrary image containing a barcode. Calculate an image profile along the barcode.

**Note.** Please note that when doing the practical assignment you

are not allowed to use the *“Lenna”* image or any other image that was used either in this book or during the presentation.

Task1:

Gray stretch

Definition: Gray scale stretch, also known as contrast stretch, is a simple linear point operation.

Function: Expand the histogram of the image to fill the whole range of gray levels.

**Formula:** g(x,y) = 255 / (B - A) \* [f(x,y) - A]

Where, A = min[f(x,y)], minimum gray level; B = max[f(x,y)], maximum gray level; f(x,y) is the input image, and g(x,y) is the output image

Disadvantages: If the minimum value A=0 and the maximum value B=255 in the gray image, the image does not change.

**Original images:**

****

**Code:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

def grey\_scale(image):

    img\_gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

    input\_min, input\_max = img\_gray.min(), img\_gray.max()

    print('input\_min = %d,input\_max = %d' % (input\_min, input\_max))

    output = np.uint8(255 / (input\_max - input\_min) \* (img\_gray - input\_min) + 0.5)

    return output

def show\_histogram(fx, gx):

    plt.figure(1)

    plt.hist(fx.ravel(), 256, [0, 256])

    plt.figure(2)

    plt.hist(gx.ravel(), 256, [0, 256])

    plt.show()

img = cv2.imread('yjsp.jpg')

greyScale = grey\_scale(img)

cv2.imshow('img', img)

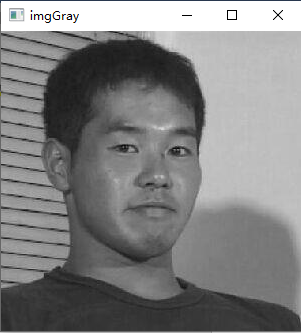
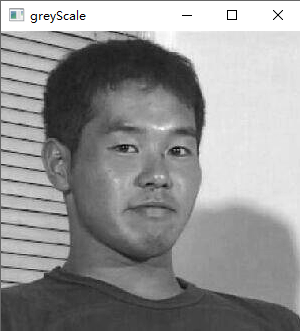
cv2.imshow('imgGray', cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY))

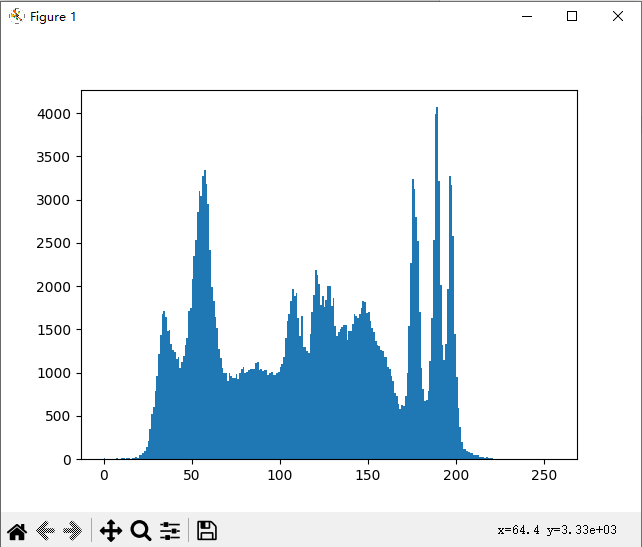
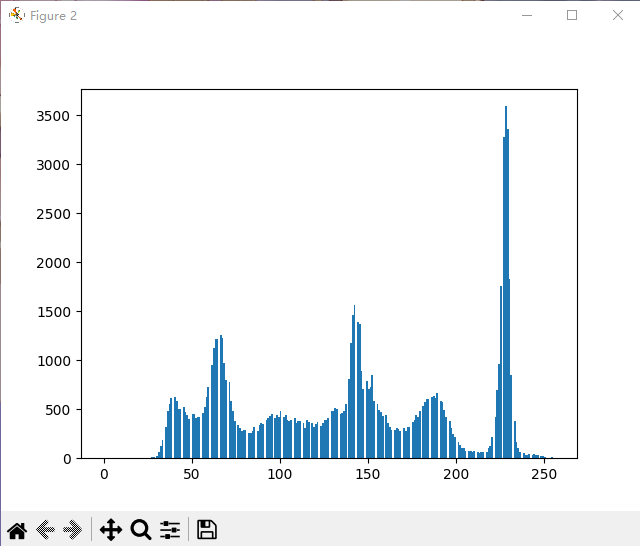
cv2.imshow('greyScale', greyScale)

show\_histogram(img, greyScale)

cv2.waitKey(0)

**Resulting images:**

** **

** **

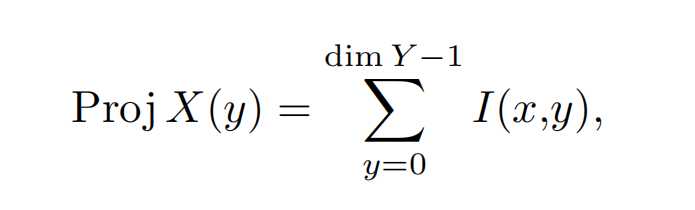
**Task2:**

**Image Projection**

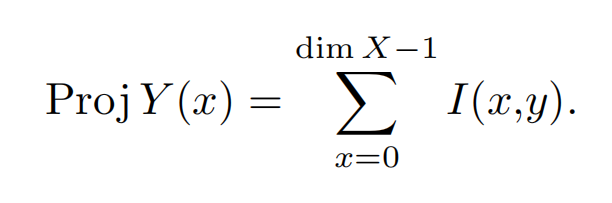
The image projection onto a certain axis is the sum of the image

pixels intensities in the direction perpendicular to this axis.

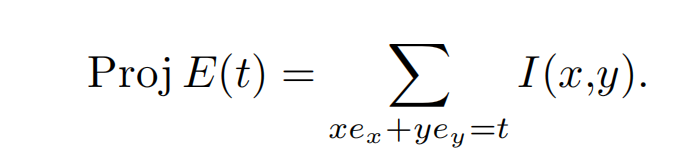
*Image Projection* onto a certain axis is the image pixels intensities sum in the direction perpendicular to this axis. The simplest case of two-dimensional image projection is the vertical projection on the axis , which is the pixel intensities sum *by columns* of the image:



where horizontal projection onto axis is the pixel intensities sum *by rows* of the image:



Let’s write an expression for the projection onto an arbitrary axis. Suppose that the axis direction is given by a unit vector with coordinates (*,*). Then the image projection onto the axis is determined by the following expression:



Projection array analysis allows you to select the projection function characteristic points, which correspond to the objects contours in the image. For example, if there are contrasting objects in the image, then the projection will see the function extremum corresponding to the each .

**Original images:**

****

**Code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def projection\_to\_vertical(img):

    projection = np.zeros(img.shape[1])

    for i in range(img.shape[1]):

        projection[i] = np.sum(img[:, i])

    return projection

def projection\_to\_horizontal(img):

    projection = np.zeros(img.shape[0])

    for i in range(img.shape[0]):

        projection[i] = np.sum(img[i, :])

    return projection

def border(projection):

    p1 = 0

    p2 = 0

    for i in range(len(projection)):

        if projection[i] - projection[i + 1] > 500 and p1 == 0:

            p1 = i

            break

    for i in range(len(projection) - 1, 0, -1):

        if projection[i] - projection[i - 1] > 500 and p2 == 0:

            p2 = i

            break

    return p1, p2

img = cv2.imread('ys.png')

ret, img = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

projection\_to\_horizontal = projection\_to\_horizontal(img)

projection\_to\_vertical = projection\_to\_vertical(img)

x1, x2 = border(projection\_to\_horizontal)

y1, y2 = border(projection\_to\_vertical)

xMax = projection\_to\_horizontal.max()

yMax = projection\_to\_vertical.max()

for i in range(len(projection\_to\_horizontal)):

    projection\_to\_horizontal[i] = -projection\_to\_horizontal[i] + xMax

for i in range(len(projection\_to\_vertical)):

    projection\_to\_vertical[i] = -projection\_to\_vertical[i] + yMax

plt.figure(1)

plt.plot(projection\_to\_horizontal)

plt.show()

plt.figure(2)

plt.plot(projection\_to\_vertical)

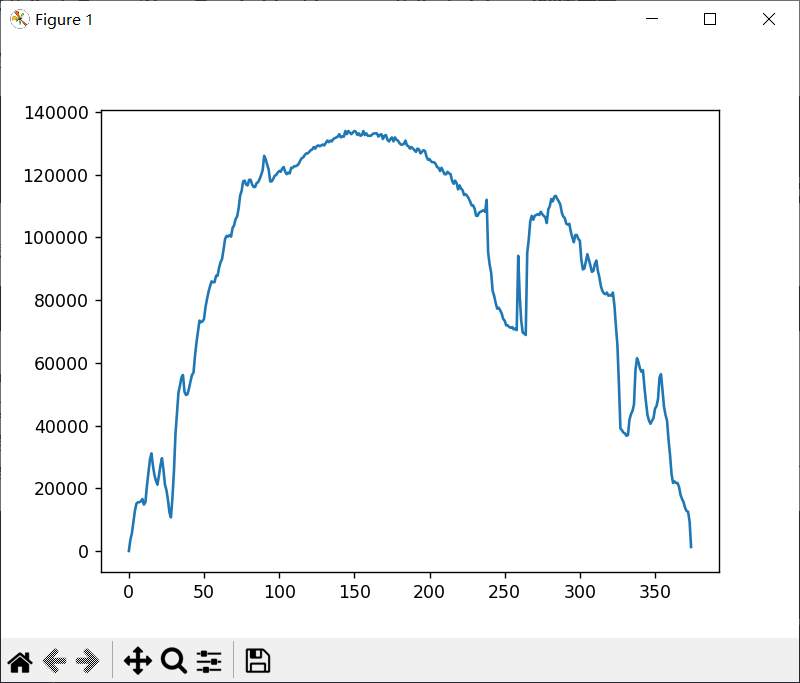
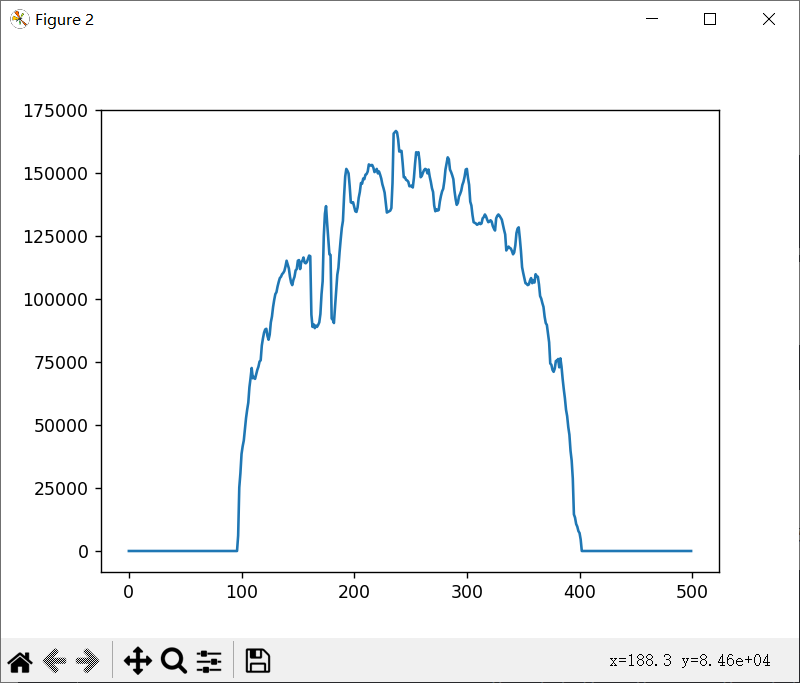
plt.show()

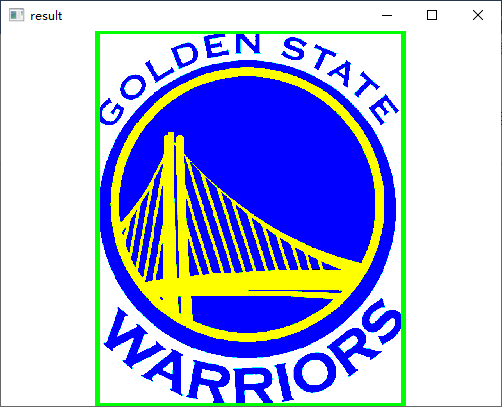
cv2.rectangle(img, (y1, x1), (y2, x2), (0, 255, 0), 4)

cv2.imshow('result', img)

cv2.waitKey(0)

**Resulting images:**

** **

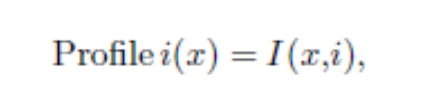
****

**Task3:**

**Image Projection**

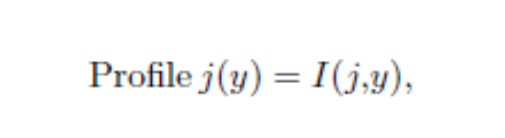
**Image Profile**

*Image profile* along an arbitrary line is the intensity function image which is distributed along this line. The simplest case of image profile is a row profile:



where — row number of the image .

Column profile is:

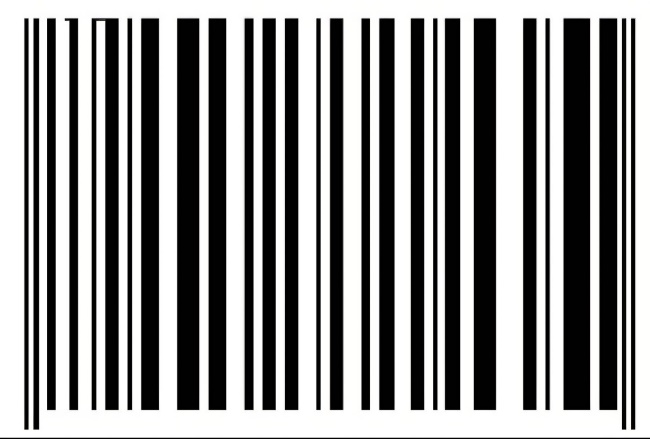


where — column number of the image .

In general case, an image profile can be considered along arbitrary straight, poly or curved line intersects the image. After image profile array getting it can be analyzed using standard approaches. The analysis allows to automatically highlight the special points of the profile function corresponding to the image contours intersected by this

line. For example, in fig. 1.1 presents the barcode image profile taken along the axis . This profile contains all the information for barcode reading, as it allows to define the interleaving sequence of «thick» and «thin» lines and spaces of a various width. In MATLAB image profile you can find using function improfile().

**Original images:**



**Code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def show\_histogram(img):

    plt.figure(1)

    plt.plot(img)

    plt.savefig('3.jpg')

    plt.show()

def projection\_to\_vertical(img):

    projection = np.zeros(img.shape[1])

    for i in range(img.shape[1]):

        projection[i] = np.sum(img[:, i])

    return projection

img = cv2.imread('txm.jpg')

ret, img = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

profile = img[round(img.shape[0] / 2), :]

profile = cv2.transpose(profile)

profile = cv2.flip(profile, 1)

projection = projection\_to\_vertical(profile)

show\_histogram(projection)

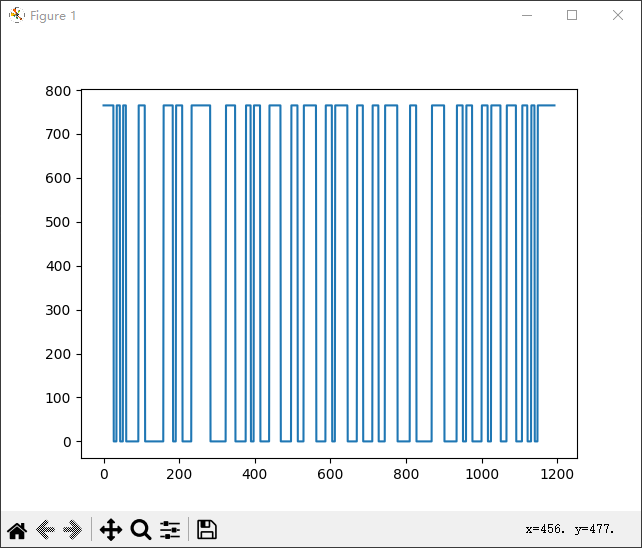
cv2.imshow("profile", profile)

cv2.imwrite('profile.jpg', profile)

cv2.waitKey(0)

**Resulting images:**





**Questions to Practical Assignment Report Defense**

1. What is the image contrast and how can you change it?

(1).Contrast is the values interval between the minimum and maximum image brightness.

(2).Use the imadjust function in matlab to increase the image contrast by original image intensity range changing.

1. Why is effective using of image profiles and projections?

In general, the image contour can be considered to intersect the image along any line, polygon or curve. After the image profile array is obtained, it can be analyzed by standard methods.Therefore, it is convenient to obtain the image profile along any axis.

1. How can you find an object against a uniform background?

The feature points of projection function are selected by projection array analysis, which correspond to the object contour in the image. The projection shows the extremum of the function corresponding to the position of each object.