机器视觉技术 第二次实验

实验题目 1——距离变换(Matlab)

实验目的：

计算二值图像中，每个像素点到最近背景像素点的距离，最后给出距离检测的斜切。

实验原理：

距离变换基于二值图像，计算其中每个像素点与其最近背景像素点之间的距离，结果为一幅灰度图像。这种变换在图像分析、计算机视觉和模式识别等领域有广泛应用。

实验步骤：

1. 创建大小为 M×N 的二维数组 F，子集 S 元素位置置为 0，其他位置置为无穷大。
2. 从上到下、从左至右遍历图像。利用上方和左侧邻接像素(AL 集合)，计算距离值。
3. 从下到上、从右至左遍历图像。利用下方和右侧邻接像素(BR 集合)，计算距离值。
4. 数组 F 的结果即为子集 S 的斜切。

程序代码：

main.m

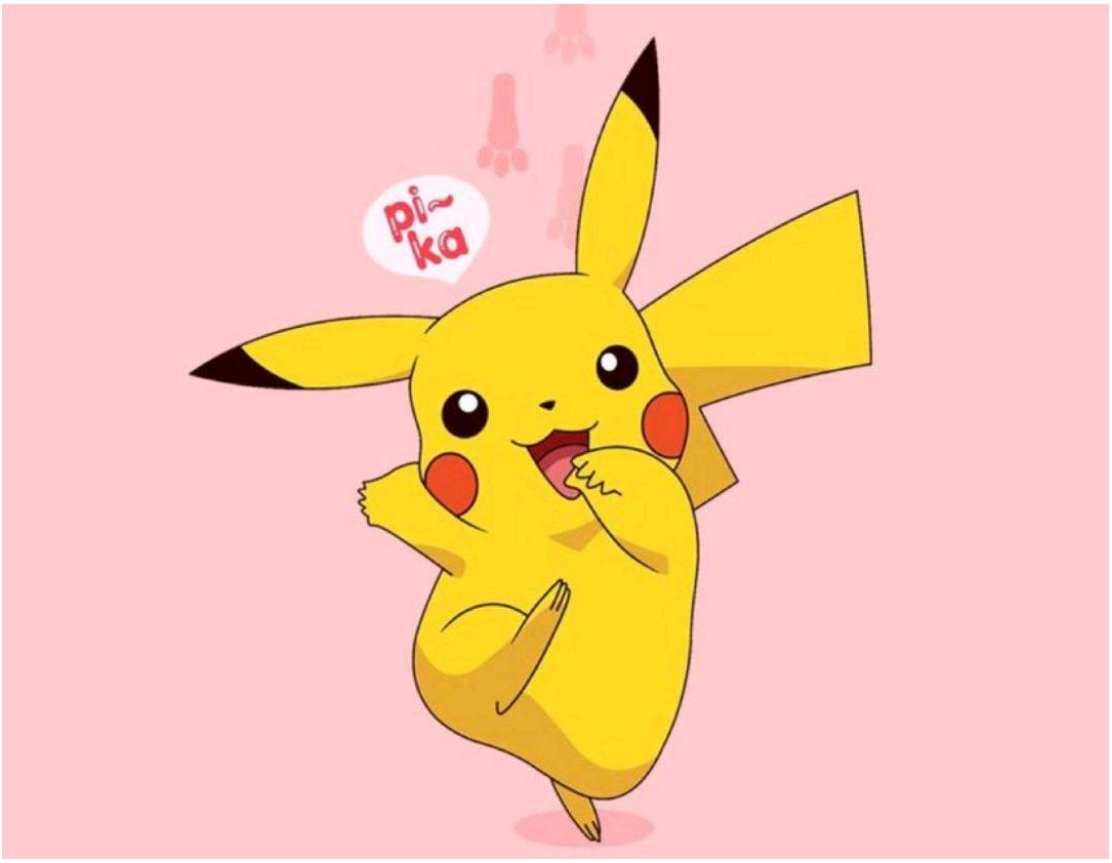
```
clc;clear all;close all;

% 读入图像
img = imread('testimg.jpg');

% 将读入的彩色图像转换为二值图像
bw = im2bw(img);

% 距离变换
DisTrans_result = MyDisTrans(bw);

% 展示结果
figure;
imshow(DisTrans_result,[]);
title('DisTrans-result');
```

MyDisTrans.m（市区距离）

```
% 定义函数 MyDisTrans，参数为二值图像 bw
function DisTrans_result=MyDisTrans(bw)
```

```
% 获取 bw 的行数 rows 和列数 cols
[rows, cols] = size(bw);

% 在 bw 周围填充 1 个像素，并用 1 填充边界，得到 bw_a，并将每个像素值乘 100
padsize = [1 1];
bw_a = padarray(bw, padsize, 1);
bw_a = bw_a * 100;

% 从(2,2)到(rows+1,cols+1)，遍历 bw_a 的每个像素
% 数组 al 包含当前像素及其左上、右上、左和上方的像素值，并根据距离加上对应值，其中最小值为当前像素值
for j = 2:cols+1
    for i = 2:rows+1
        al = [bw_a(i,j) bw_a(i-1,j-1)+2 bw_a(i+1,j-1)+2 bw_a(i,j-1)+1 bw_a(i-1,j)+1];
        bw_a(i,j) = min(al);
    end
end

% 反向再遍历一次
for j = cols+1:-1:2
    for i = rows+1:-1:2
        bl = [bw_a(i,j) bw_a(i,j+1)+1 bw_a(i+1,j+1)+2 bw_a(i-1,j+1)+2 bw_a(i+1,j)+1];
        bw_a(i,j) = min(bl);
    end
end

% 返回 bw_a 的中间部分
DisTrans_result = bw_a(2:rows+1,2:cols+1);

end
```

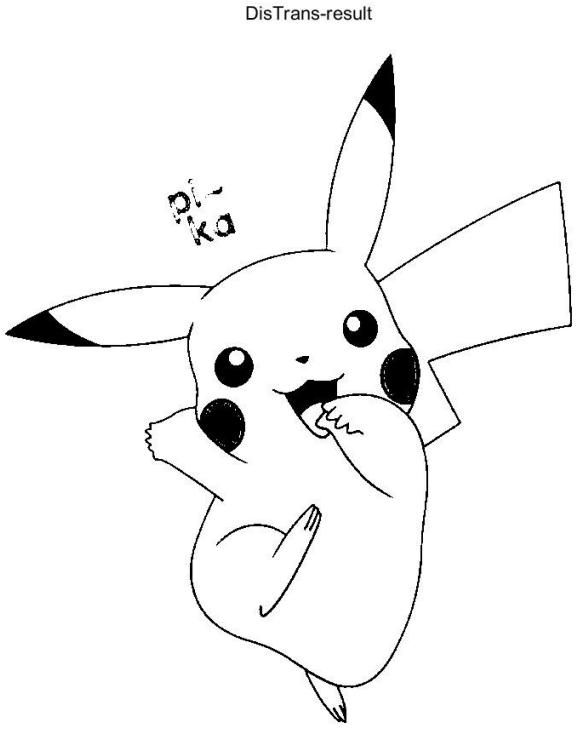
实验结果显示：
测试图片：



结果 1（棋盘距离）：

结果 2（市区距离）：

实验分析总结：
以下为 Matlab 用 bwdist(i, '')函数的测试结果
结果 1（棋盘距离）：



结果 2（市区距离）：

DisTrans-result

通过比较函数和代码的两种距离方式的结果，可以得到以下结论：

1、棋盘距离所加值为 1，会更加模糊，而市区距离所加值为 1 或 2，能更清晰地看出原图轮廓。

2、直接调用的函数的效果并不明显，这可能与代码中“`bw_a = bw_a * 100;`”这一句有关。

实验题目 2——直方图显示(C++)

实验目的：

给定一张图片，将其灰度化处理后，统计像素的亮度情况，绘制亮度直方图。

实验原理：

直方图可以展示数据的分布，而亮度直方图展示了图像中不同亮度级别的像素的分布情况。其横轴通常代表亮度级别，为 0-255 的整数，0 代表黑色，255 代表白色；而纵轴则代表每个亮度级别对应的像素数量或频率。通过亮度直方图，可以观察图像的亮度分布、曝光、整体亮度水平等信息，这对于图像调整、增强和后期处理非常有用。

在制作亮度直方图时，首先初始化长度为 255 的一维数组，其代表 256 个亮度级别，并将数组全部置零。遍历图像的像素，根据其亮度级别，在对应的位置加 1。在遍历全图后，即得到亮度直方图。

实验步骤：

1. 创建大小为 k 的一维数组 H。
2. 将数组 H 的所有元素初始化为 0。

3. 对于图像的每一个元素，做如下处理：H[f(m,n)] = H[f(m,n)]+1。

程序代码：

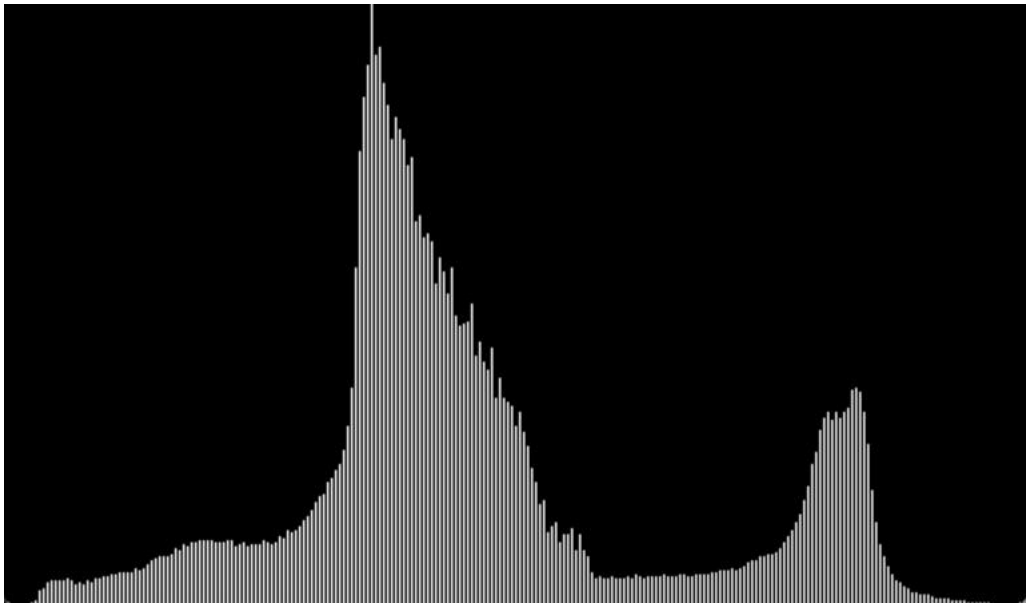
```
#include <iostream>
#include <fstream>
#include "opencv2/opencv.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdio.h>
using namespace cv;
using namespace std;

Mat myHist(Mat img)
{
    // 定义通道数组，只对0通道（灰度图像）进行操作
    int channels[] = { 0 };
    // 定义直方图大小为256，对应0-255的灰度值，并将范围转化为指针
    int histSize[] = { 256 };
    float range[] = { 0,255 };
    const float* pRange[] = { range };

    // 创建Mat对象存储直方图结果，输入图像为img，通道为channels，不使用掩膜，直方图结果存储在Hist中
    Mat Hist;
    calcHist(&img, 1, channels, Mat(), Hist, 1, histSize, pRange);

    // 获取直方图的最大值和最小值，并构建绘制的图像
    double max_val, min_val;
    minMaxLoc(Hist, &min_val, &max_val, 0, 0);
    Mat histImg(Size(256 * 2, 300), CV_8U, Scalar(0));

    // 遍历直方图，得到各灰度值的数值
    for (int i = 0; i < histSize[0]; i++) {
        int val = int(Hist.at<float>(i) / max_val * 300);
        line(histImg, Point(i * 2, 300), Point(i * 2, 300 - val), Scalar(255, 255, 255),1);
    }

    return histImg;
}

void main()
{
    Mat input = imread("testimgl.jpg");
```

```
    //彩色图转为灰度图
    Mat gray;
    cvtColor(input, gray, COLOR_BGR2GRAY);

    //直方图绘制
    Mat Hist = myHist(gray);

    imshow("Hist", Hist);
    waitKey(0);
}
```

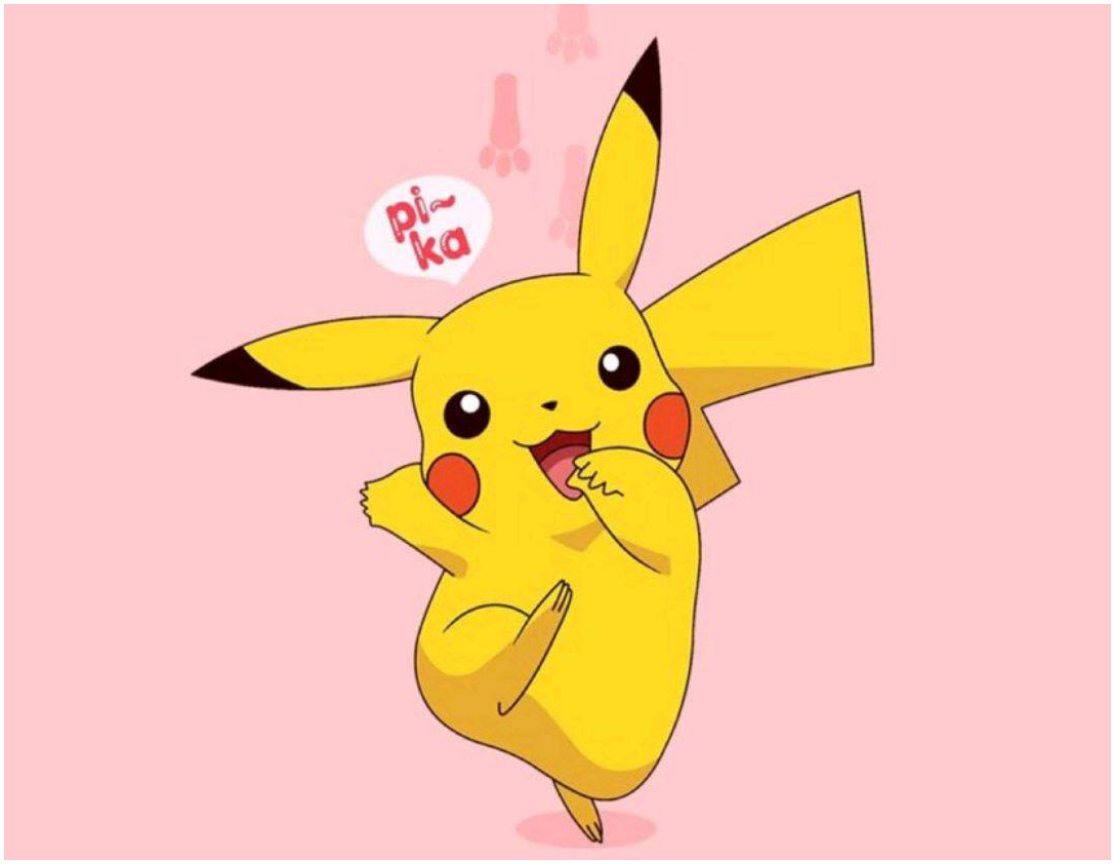
实验结果显示：

测试图片 1：

结果 1：



测试图片 2：

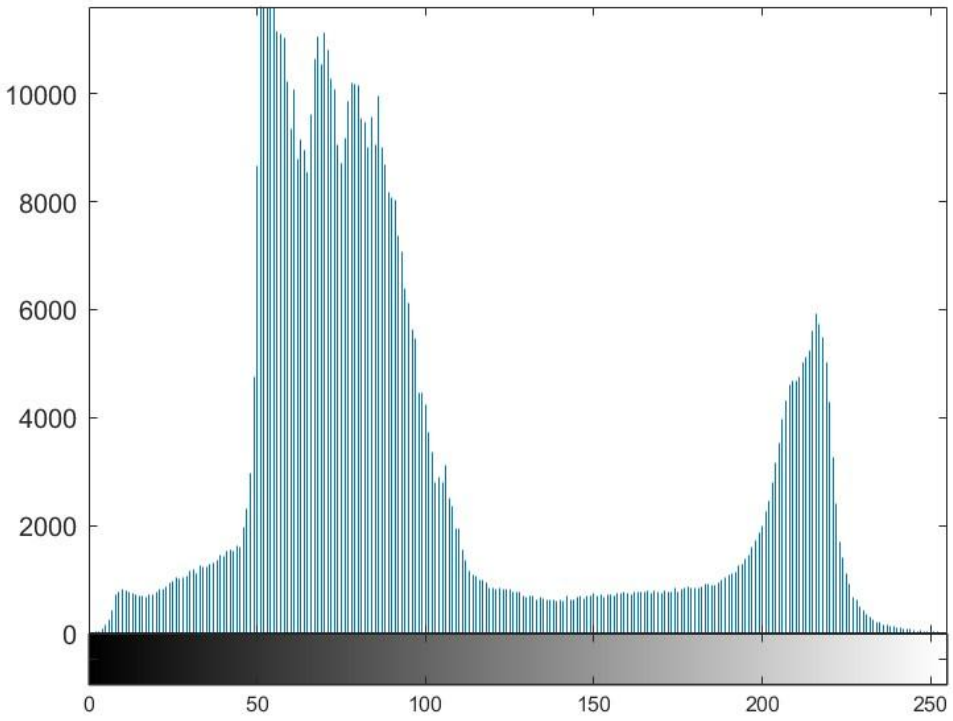


结果 2：

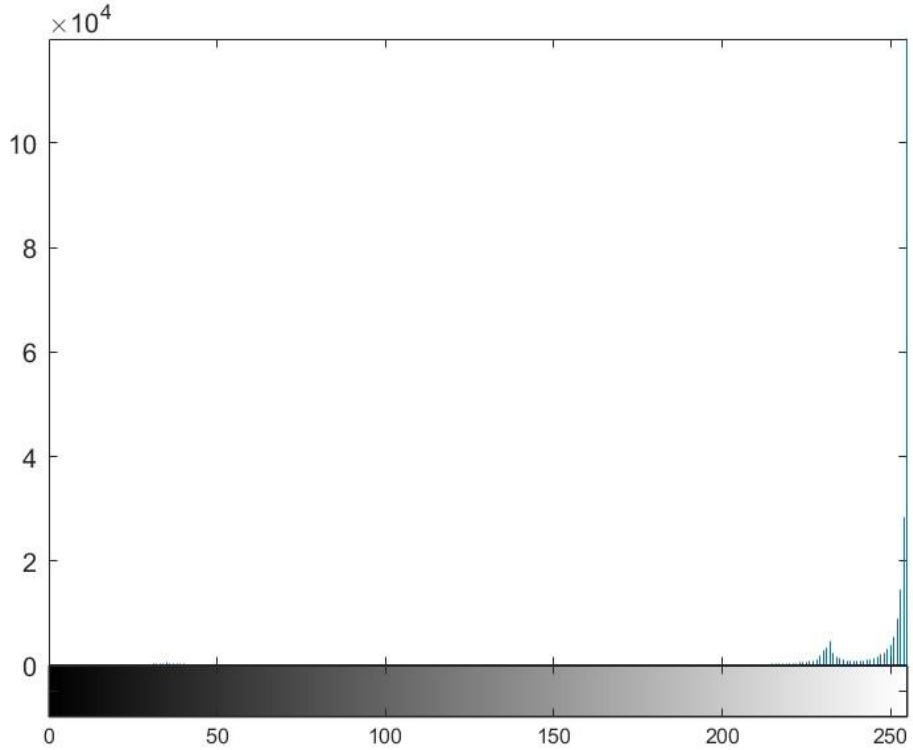
实验分析总结：

以下为 Matlab 用 （i(:,:,1)）函数的测试结果

结果 1：



结果 2：



通过比较函数和代码的两种直方图的结果，可以得到以下结论：二者的结果基本一致，但在绘图上有所不同。Matlab 的函数限制了横轴的范围为 0-255，而 C++中，虽然规定了范围，但是图表仍向后延申。