

机器视觉技术 第六次实验

实验题目 1——p 率阈值化

实验目的：

基于图像直方图选择阈值 T ，使得 $1/p$ 的图像面积具有比 T 小的灰度值。

实验原理：

根据图像直方图得到累积直方图，再由此确定一个阈值，使图像中灰度值小于该阈值的像素面积占总图像面积的 $1/p$ 。

实验步骤：

1. 获得图像的直方图，再得到累积直方图，由此确定阈值 T 。
2. 使图像中灰度值小于 T 的像素面积占总图像面积的 $1/p$ 。

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 myThresholdP(Mat img)
{
    Mat Hist;

    // 计算直方图
    int histSize = 256;
    float range[] = { 0, 256 };
    const float* histRange = { range };
    bool uniform = true, accumulate = false;
    calcHist(&img, 1, 0, Mat(), Hist, 1, &histSize, &histRange, uniform, accumulate);

    float P = 0.5; //设定P值
    float T = 0; //经由直方图得到的阈值
    float sum_pixels = 0;
    for (int i = 0; i < histSize; i++) {
        sum_pixels += Hist.at<float>(i);
        if (sum_pixels >= img.rows * img.cols * P) {
            break;
        }
    }
```

```

        T = i;
    }

    // 根据T对图像进行二值化处理
    Mat ThresholdPImg = img.clone();
    for (int i = 0; i < img.rows; i++) {
        for (int j = 0; j < img.cols; j++) {
            if (ThresholdPImg.at<uchar>(i, j) >= T) {
                ThresholdPImg.at<uchar>(i, j) = 255;
            }
            else {
                ThresholdPImg.at<uchar>(i, j) = 0;
            }
        }
    }

    //返回p率阈值化结果
    return ThresholdPImg;
}

void main()
{
    Mat input = imread("testing.jpg");
    Mat gray;
    cvtColor(input, gray, COLOR_BGR2GRAY);

    //灰度图p率阈值化
    Mat ThresholdPImg = myThresholdP(gray);

    imshow("ThresholdPImg", ThresholdPImg);
    waitKey(0);
}

```

实验结果显示：

测试
图片



P=
0.
5



P=
0.
75





实验分析总结：

p 率阈值化很好地按照像素的灰度值对像素进行了分类，可以对比突出图像的前景和背景。观察对比上述三个不同 p 值对应的测试结果，可以发现其很好地与理论相合，p 值从小到大，黑色部分也从少到多。

但是 p 率阈值化也有一定问题。一个是 p 值的确定。对一幅图片来说，很难把控区分前景和背景的灰度值阈值，p 值需要反复试验获得。由此引发第二个问题，一张图片的不同区域，可能具有不同的阈值，很难周全地划分。如本测试图片的结果中，不同的 p 值都会不同层度地把部分天空设成黑色，这是不完美的。

实验题目 2——迭代的最优阈值选择

实验目的：

通过迭代，不断逼近划分前景和背景的最优亮度阈值。

实验原理：

从图像已有像素亮度的中值出发，将图像分成两个部分，不断迭代获得新的阈值，使图像的前景和背景能被明显区分。当阈值的改变小于一定值时，视为获得正确的阈值，并将前景赋为黑色，背景赋为白色。

实验步骤：

1. 设定初始估计阈值 $t=t_0$ （最大亮度值与最小亮度值的中间值）。 t_1 为迭代中变化得到的新阈值，由数组 t 储存。
2. 使用阈值 t_0 分割图像，获得两部分图像，求各自的平均灰度值， t_1 为两个平局值的平均数。
3. 重复步骤 2，直到 $|t_0-t_1|$ 的变化足够小，本实验中设定为 1。
4. 输出处理后的图像和 t 的变化曲线。

程序代码：

main.m

```

clc;clear all;close all;

% 读入图像并转为灰度图
img=imread('testimg.jpg');
img=rgb2gray(img);

% 调用函数
[OptimalThreshold_result,t_changing] = MyOptimalThreshold(img);

% 处理后的图像
figure;
imshow(OptimalThreshold_result);
title('OptimalThreshold_result');

% t 的变化曲线
figure;
plot(1:length(t_changing), t_changing);
xlabel('times');
ylabel('t');
title('t_changing');

MyOptimalThreshold.m
% 定义函数 MyOptimalThreshold, 参数为二值图像 img
function [OptimalThreshold_result,t_changing]=MyOptimalThreshold(img)

img = double(img);

% 初始化与存储数组
t0 = 0;
t1 = (min(img(:)) + max(img(:))) / 2;
t = zeros(1, 100);
t_index = 1; % 当前阈值在数组中的索引

% 迭代过程
while abs(t0 - t1) > 1
    t0 = t1;
    img1 = img >= t0;
    img2 = img < t0;
    u1 = mean(img(img1));
    u2 = mean(img(img2));
    t1 = (u1 + u2) / 2;
    t(t_index) = t1;
    t_index = t_index + 1;
end

```

```

% 结果图像
new_img = zeros(size(img));
new_img(img >= t1) = 0;
new_img(img < t1) = 1;





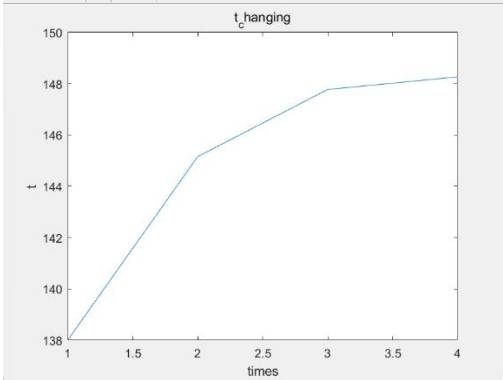
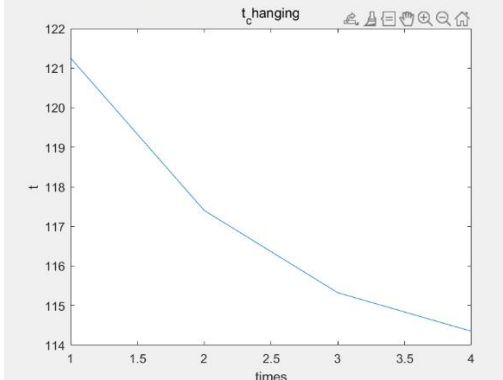
t = t(1:t_index-1);

% 返回值
OptimalThreshold_result=new_img;
t_changing=t;

end

```

实验结果显示：

项目	测试组 1	测试组 2
测试图片		
处理结果		
t 的变化曲线		

实验分析总结：

以上测试了两组照片，可以看出，教学楼的亮度对比要弱于 Lenna 女士的照片，二者的阈值 t 的取值也相差不少。该迭代处理方法效果显著，可以看见，教学楼的窗户的轮廓都被标注了出来。观察 t 的迭代曲线，可以发现，在迭代过程中， t 的变化速率逐渐减缓，最后缓慢逼近准确值，这也是符合原理的。与 p 率阈值化的方法相对比，本方法在区分前景和背景方面可能更好，因为可以自动寻找最合适的阈值。

本次实验中，学习使用了 `matlab` 中的多个常用函数，了解了它们的用法，这省去了众多循环、判断语句，十分的方便。