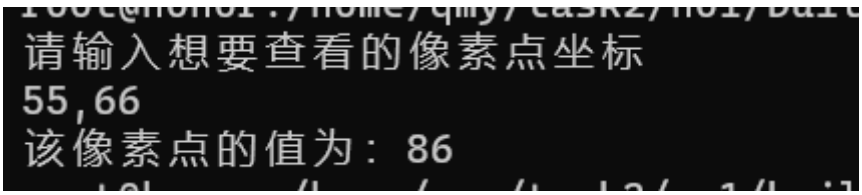
第二次任务

任务一

操作每个像素点

- 以单通道方式打开图片

- 查看任意像素点



- 每个像素点加50——变亮

- 每个像素点减80——变暗

代码1

```
#include <iostream>
#include "opencv2/opencv.hpp"
using namespace std;
using namespace cv;
int main()
{
    Mat img = imread("1.jpg", 0); //以单通道读入图片
    for(int i=0;i<234;i++)
    {
        for(int j=0;j<417;j++)
        {
```

```
            img.at<uchar>(i, j) +=50;
        }
    }
    imshow("图像", img);
    waitKey(0);
    return 0;
}
```

RGB转HSV

- 转化公式

  1. RGB2HSV

  将图像由RGB色彩空间转换为HSV色彩空间时，处理方式如下：

$$V = max(R, G, B)$$

$$S = \begin{cases} \frac{V - min(R,G,B)}{V} & V \neq 0 \\ 0 & \text{其他} \end{cases}$$

$$H = \begin{cases} \frac{60(G-B)}{V-min(R,G,B)} & V = R \\ 120 + \frac{60(B-R)}{V-min(R,G,B)} & V = G \\ 240 + \frac{60(R-G)}{V-min(R,G,B)} & V = B \end{cases}$$

  如果结果存在的情况，进一步计算：

$$H = \begin{cases} H + 360 & H < 0 \\ H & \text{其他} \end{cases}$$

  由上述公式计算后：

$$S \in [0, 1]$$
$$V \in [0, 1]$$
$$H \in [0, 360]$$

- 思路，编写一个函数，将RGB转为HSV

- 展示结果 

代码2

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace std;
using namespace cv;
//RGB颜色结构体
struct RGBColor
{
    int R;
    int G;
    int B;
};

//HSV颜色结构体
struct HSVColor
{
    double H;
    double S;
    double V;
};

// 将RGB颜色转换为HSV颜色
HSVColor RGBtoHSV(const RGBColor& rgb) {
    HSVColor hsv;

    double R = static_cast<double>(rgb.R) / 255.0;
    double G = static_cast<double>(rgb.G) / 255.0;
```

```
    double B = static_cast<double>(rgb.B) / 255.0;

    double Cmax = std::max(R, std::max(G, B));
    double Cmin = std::min(R, std::min(G, B));
    double delta = Cmax - Cmin;

    // 计算H（色调）
    if (delta == 0) {
        hsv.H = 0.0;  // 无色
    }
    else if (Cmax == R)
    {
        hsv.H = 60.0 * fmod((G - B) / delta, 6.0);
    }
    else if (Cmax == G)
    {
        hsv.H = 60.0 * ((B - R) / delta + 2.0);
    }
    else if (Cmax == B)
    {
        hsv.H = 60.0 * ((R - G) / delta + 4.0);
    }

    // 计算S（饱和度）
    if (Cmax == 0)
    {
        hsv.S = 0.0;
    }
    else
    {
        hsv.S = delta / Cmax;
    }

    // 计算V（亮度）
    hsv.V = Cmax;

    return hsv;
}

int main()
{
    Mat image = imread("/home/qmy/task2/no1/2.jpg");
    // 将RGB图像转换为HSV图像
```

```
    Mat hsv_image(image.size(), CV_8UC3);
    for (int y = 0; y < image.rows; y++)
    {
        for (int x = 0; x < image.cols; x++)
        {
            Vec3b rgb = image.at<Vec3b>(y, x);
            RGBColor rgb_color = { rgb[2], rgb[1], rgb[0] };
            HSVColor hsv_color = RGBtoHSV(rgb_color);
            hsv_image.at<Vec3b>(y, x) = Vec3b(hsv_color.H / 2, hsv_color.S * 255, hsv_color.
        }
    }

    // 显示HSV图像
    imshow("HSV结果", hsv_image);
    waitKey(0);
    return 0;
}
```

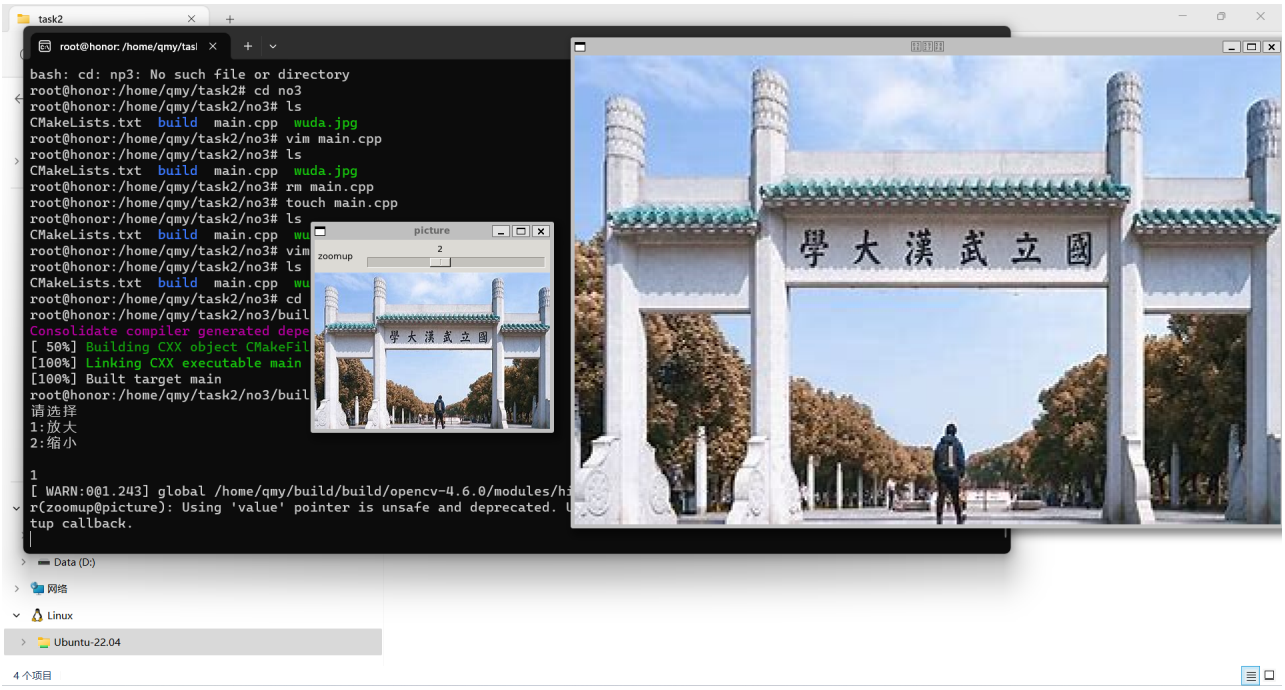
任务二

思路

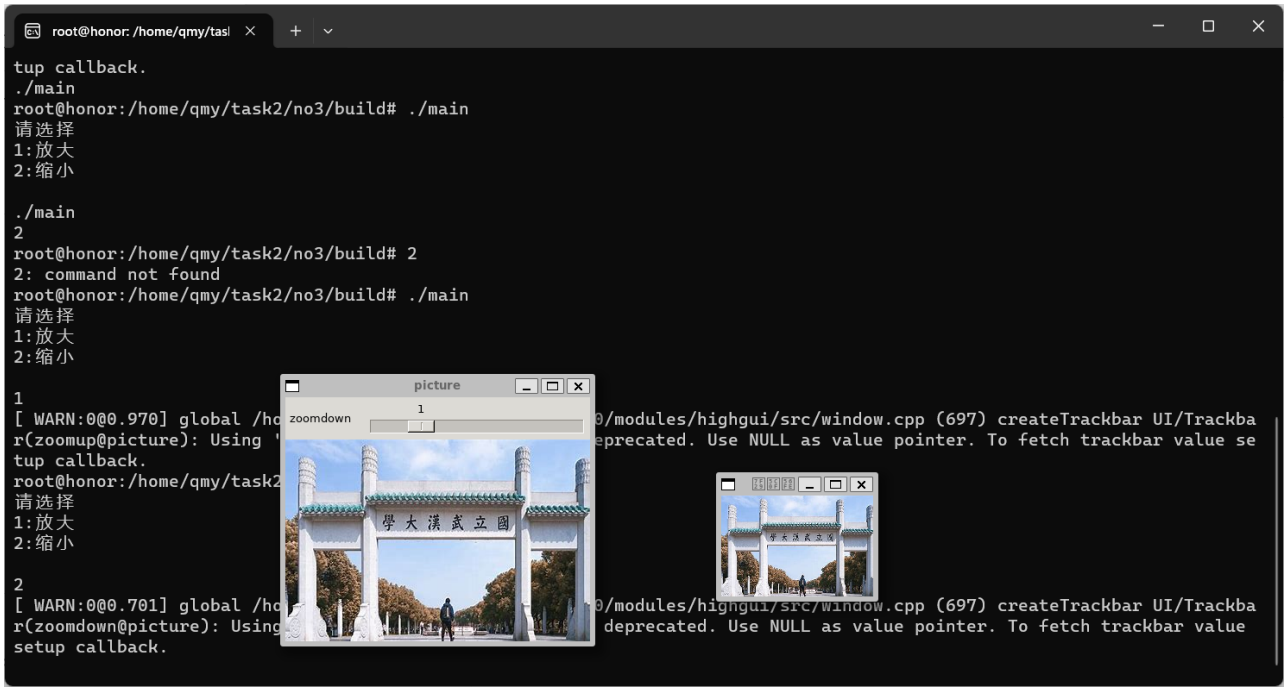
以原图片格式创建一个大小对应缩放系数的图像—用at方法将每个像素值乘以/除以系数来达到放大或缩小的效果

结果

- 放大



- 缩小



代码

```
#include <opencv2/opencv.hpp>
#include <vector>
using namespace std;
using namespace cv;

void zoomup(int k, void* usrdata)
{
    Mat img = *(Mat*)(usrdata);
    Mat up(img.rows * (k+1), img.cols * (k+1), img.type());//k+1是因为滑动条到0会报错退出
    // 使用at方法执行向上采样
    for (int y = 0; y < up.rows; y++)
    {
        for (int x = 0; x < up.cols; x++)
        {
            up.at<Vec3b>(y, x) = img.at<Vec3b>(y / (k + 1), x / (k + 1));//将三通道img每个
        }
    }
    imshow("放大图", up);
}
void zoomdown(int k, void* usrdata)
{
    Mat img = *(Mat*)(usrdata);
    Mat down(img.rows / (k + 1), img.cols / (k + 1), img.type());
    // 使用at方法执行向上采样
    for (int y = 0; y < down.rows; y++)
```

```
    {
        for (int x = 0; x < down.cols; x++)
        {
            down.at<Vec3b>(y, x) = img.at<Vec3b>(y * (k + 1), x * (k + 1));//将三通道img每个
        }
    }
    // 显示向上采样后的图像
    imshow("缩小图", down);
}
int main()
{
    int flag = 0;
    cout << "请选择\n1:放大\n2:缩小\n" << endl;
    cin >> flag;
    int  k = 0;
    int max = 5;
    Mat img = imread("../wuda.jpg");
    namedWindow("picture");
    imshow("picture", img);
    if (flag == 1)
        createTrackbar("zoomup", "picture", &k, max, zoomup, &img);
    else if(flag==2)
        createTrackbar("zoomdown", "picture", &k, max, zoomdown, &img);
    waitKey(0);
    destroyAllWindows();
    return 0;
}
```

任务三

思路

读取图像—提取车牌蓝色的HSV值—利用函数提取图像中HSV值为蓝色的范围—转化为蓝色为白，其他颜色为黑的二值化图像(用到了膨胀操作，目的是消除车牌中文字的干扰）-外接矩形提取轮廓—保存此轮廓

结果展示

准确提取了轮廓

轮廓保存为图片

代码

```
#include<opencv2/opencv.hpp>
#include <iostream>
using namespace std;
using namespace cv;

// 颜色识别识别车牌颜色
Mat ColorFindContours(Mat srcImage,
    int iLowH, int iHighH,
    int iLowS, int iHighS,
    int iLowV, int iHighV)
{
    Mat bufImg;
    Mat imgHSV;
    //转为HSV
    cvtColor(srcImage, imgHSV, COLOR_BGR2HSV);
    inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), bufImg);
    return bufImg;
}
```

```
Mat ColorFindContours1(Mat srcImage)
{
    Mat des1 = ColorFindContours(srcImage,
        200/2, 248/2 ,                // 色调最小值~最大值
        int(255*0.85), int(255),   // 饱和度最小值~最大值
        (255*0.68), (255*0.90));  // 亮度最小值~最大值

    return des1;
}
Mat car(Mat img)
{
    Mat des2 = ColorFindContours1(img);
    Mat binary;
    Mat dst;
    GaussianBlur(des2, des2, Size(9, 9), 2, 2);//高斯滤波
    threshold(des2, binary, 170, 255, THRESH_BINARY | THRESH_OTSU);//二值化

    Mat kernel = getStructuringElement(MORPH_RECT, Size(22, 22), Point(-1, -1));//膨胀操作参
    dilate(binary, dst, kernel);

    vector<vector<Point>> contours;
    vector<Vec4i> hierarchy;
    findContours(dst, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE, Point());

    Point2f ps[4];
    for (int i = 0; i < contours.size(); ++i)
    {  //遍历所有轮廓
        double areal = contourArea(contours[i]);
        if (areal > 100)
        {
            Rect rect = boundingRect(contours[i]);  //获取轮廓的外接矩形
            rectangle(img, rect, Scalar(0,255,0), 5);  //绘制轮廓矩形
            //保存轮廓
            Mat selectedRegion = img(rect);
            imwrite("jieguo.jpg", selectedRegion);
        }
    }
    return img;
}
int main()
{
```

```
    Mat img = imread("/home/qmy/task2/no2/3.png");
    Mat result = car(img);
    namedWindow("result", 0);
    imshow("result", result);//显示结果
    waitKey(0);


    return 0;
}
```