Opencv

提示：回车即可创建新的条目， `tab` *即可缩进为子项，* `shift` + `tab` *即可恢复为父项*
打开右上角 ••• 菜单，选择“全局选项”—>“辅助提示”中的“显示块结构”，可以更好地区分层级

- 画图形
  - resize（src,dst,dsize,fx,fy,interpolation）缩放图像函数

    src：输入，原图像，即待改变大小的图像；

    dst：输出，改变大小之后的图像，这个图像和原图像具有相同的内容，只是大小和原图像不一样而已；

    dsize：输出图像的大小。如果这个参数不为 0，那么就代表将原图像缩放到这个 Size(width，height)指定的大小；如果这个参数为 0，那么原图像缩放之后的大小就要通过下面的公式来计算：dsize = Size(round(fx*src.cols), round(fy*src.rows))；其中，fx 和 fy 就是下面要说的两个参数，是图像 width 方向和 height 方向的缩放比例。

    fx：width 方向的缩放比例，如果它是 0，那么它就会按照(double)dsize.width/src.cols 来计算；

    fy：height 方向的缩放比例，如果它是 0，那么它就会按照(double)dsize.height/src.rows 来计算；

    interpolation：这个是指定插值的方式，图像缩放之后，肯定像素要进行重新计算的，就靠这个参数来指定重新计算像素的方式，有以下几种：
  - Mat img;img(Rect &roi)；图像自带裁剪函数
    - int cropValue = 5;
      Rect roi(cropValue, cropValue, w – 2*cropValue, h –2* cropValue); //裁剪区域的左上点 x，y，裁剪区域宽，高
      imgCrop = imgWarp(roi);
  - Mat img(512, 512, CV_8UC3, Scalar(255, 255, 255));创建画纸
    - 宽，高，（8–8 个 bit；3–BGR 三个通道）,底色
  - circle(img, Point(256, 256), 155, Scalar(255, 0, 255),2);画圆
    circle(img, Point(256, 256), 155, Scalar(255, 0, 0), FILLED);
    - 画纸，圆心，半径，画笔颜色，线条厚度（FILLED 为填满内部）
  - rectangle(img, Point(130, 226), Point(382, 286), Scalar(255, 255, 255), FILLED);
    - 画纸，左上点，右下点，颜色，厚度
  - line(img, Point(130, 296), Point(382, 296), Scalar(255, 255, 255),2);
    - 画纸，起点，终点，颜色，厚度
  - putText(img, "Pan Zijian's Workshop", Point(137, 262), FONT_HERSHEY_DUPLEX, 0.63, Scalar(0, 255, 0), 2);
    - 画纸，文字内容，开始写的点，字体，字体大小，字体颜色，字体厚度
- 基本功能（图片预处理）
  - cvtColor(img, imgGray, COLOR_BGR2GRAY) 转为灰度图
    - COLOR_BGR2GRAY 灰度图
  - GaussianBlur(imgGray, imgBlur, Size(7, 7), 3, 0) 高斯模糊
    - 前三个数值越大越模糊
  - Canny(imgBlur, imgCann, 25,75) 边缘检测
    - 数值越小，边越细
  - dilate(imgCann, imgDil, kernel) 膨胀处理
    - Mat kernel = getStructuringElement(MORPH_RECT,Size(3,3));
      - 数值越大，边越粗,只能用奇数
  - erode(imgDil, imgErode, kernel) 腐蚀处理
    - Mat kernel = getStructuringElement(MORPH_RECT,Size(3,3));
      - 数值越大，边越粗,只能用奇数

- 颜色检测
  - cvtColor(img, imgHSV, COLOR_BGR2HSV) 转为 HSV
    - H 色调（色彩），0–179；S 饱和度（深浅）S=0 时，只有灰度，0–255；V 明度（明暗），0–255
  - namedWindow("Trackbars", (640, 200)) 创建新窗口
    - 窗口名称，窗口大小
  - 创建轨迹栏 createTrackbar("Hue Min", "Trackbars", &hmin, 179)
    createTrackbar("Hue Max", "Trackbars", &hmax, 179);
    - 头名称，要放入的窗口名称，关联的变量（通过窗口拖拉轨迹栏改变变量的值）,设置变量最大值
  - inRange(imgHSV, lower, upper, mask) 二值化处理：如果一幅灰度图像的某个像素的灰度值在指定的高、低阈值范围之内，则在 dst 图像中令该像素值为 255（黑色），否则令其为 0（白色），这样就生成了一幅二值化的输出图像。
    - Scalar lower(hmin, smin, vmin);Scalar upper(hmax, smax, vmax);
- 人脸检测
  - 头文件#include<opencv2/objdetect.hpp>
  - CascadeClassifier faceCascade;//创建用作目标检测的级联分类器 faceCascade
  - faceCascade.load("resources/haarcascade_frontalface_default.xml");载入训练好的模型文件
  - if (faceCascade.empty()) { cout << "XML file not loaded"<<endl; };//检测文件是否已被打开
  - faceCascade.detectMultiScale(img,faces,1.1,10);人脸检测函数
    - vector<Rect> faces;//储存矩形框
    1. const Mat& image：输入图像
    2. vector& objects：输出的矩形向量组；vector<Rect> faces;//储存矩形框
    3. double scaleFactor=1.1：这个是每次缩小图像的比例，默认是 1.1
    4. minNeighbors=3：匹配成功所需要的周围矩形框的数目，每一个特征匹配到的区域都是一个矩形框，只有多个矩形框同时存在的时候，才认为是匹配成功，比如人脸，这个默认值是 3。
    5. flags=0：可以取如下这些值：

       CASCADE_DO_CANNY_PRUNING=1, 利用 canny 边缘检测来排除一些边缘很少或者很多的图像区域

       CASCADE_SCALE_IMAGE=2, 正常比例检测

       CASCADE_FIND_BIGGEST_OBJECT=4, 只检测最大的物体

       CASCADE_DO_ROUGH_SEARCH=8 初略的检测
    6. minObjectSize maxObjectSize：匹配物体的大小范围
  - rectangle(img, Rect(faces[i].tl(), faces[i].br()), Scalar(255, 0, 255), 2);//画矩形框
- 获取轮廓
  - findContours(imgDil, contour, hierarchy,RETR_EXTERNAL, CHAIN_APPROX_SIMPLE)
    - vector<vector<Point>> contour;用来储存图片（经过预处理）里面的轮廓 ；vector<Vec4i>hierarchy;四维向量，与 contour 中的元素一一对应
  - drawContours(img, contour, -1, Scalar(255, 0, 255), 2);
    - 第二个参数为储存了轮廓的容器（vector<vector<Point>>类型），第三个参数为要画出的轮廓在 contour 里的下标，-1代表描出 contour 里所有的轮廓

//至此已经可以画出轮廓，以下为框出不规则图形的实现

- approxPolyDP(contour[i], conPoly[i], 0.02*peri, true);//多边拟合函数，将图形轮廓转为多边形
  - InputArray curve:一般是由图像的轮廓点组成的点集
  - OutputArray approxCurve：表示输出的多边形点集；vector<vector<Point>> conPoly(contour.size())//用来储存多边形轮廓
  - double epsilon：主要表示输出的精度，就是另个轮廓点之间最大距离数，5,6,7，，8...参数越小，越逼近曲线
    - float peri = arcLength(contour[i], true);//弧长
    - arcLength(contour[i], true)，获取轮廓的弧长

      InputArray curve：表示图像的轮廓

      bool closed：表示轮廓是否封闭的
  - bool closed：表示输出的多边形是否封闭

- rectangle(img, boundRect[i].tl(),boundRect[i].br(), Scalar(255, 0, 0), 5);//根据左上与右下顶点画出边框矩形
    - boundRect[i].tl()：矩形左上点；boundRect[i].br()：矩形右下点
    - boundRect[i] = boundingRect(conPoly[i]);
        - boundingRect(conPoly[i]);计算多边形轮廓的垂直边界最小矩形，矩形是与图像上下边界平行的
        - vector<Rect> boundRect(contour.size());//用来储存框物体的矩形边框

即先将轮廓转为多边形框，再转为矩形框，再画框

- //做文字标记
  putText(img, objType, Point(boundRect[i].x, boundRect[i].y-5),
  FONT_HERSHEY_PLAIN,0.75,Scalar(0,0,255), 1);
  //boundRect[i].x 表示 boundRect[i]这个矩形左上点的 x 坐标

- Project1_Virtual_Paint
    - vector<vector<int>> findColors(Mat img);//1.利用 inRange 函数获得二值化图片 mask，每一个循环代表寻找一种颜色的物体 2.返回储存了每种颜色的一个物体框的中点 x,y 坐标的的二维数组，外部用 vector<vector<int>>newPoints;接收
        - Point getContours(Mat mask);//寻找某种颜色物体的轮廓（面积大于固定值），返回最后一个物体的物体框的中点（初始化为 Point myPoint(0, 0)）;，外部用 Point myPoint=getContours(mask);接收

      //newPoints.push_back({ myPoint.x,myPoint.y,i })

      每一次循环的 i 值捆绑了该次循环的二值化参数和与二值化参数对应的之后画点的颜色参数
    - void drawOnCanvas(vector<vector<int>>newPoints, vector<Scalar> myScalarValues)//在画布上作画
        - vector<vector<int>> myColors = { {0,123,149,179,255,220}//红色,{116,25,128,138,70,216//紫色} };
            - 参数具体值用自定义的 PickColor 程序获取，原理是人工拖拽轨迹栏，改变 HSV 的值，使该颜色的物体为白色，图片其余部分为黑色，并输出当前 HSV 的值
        - vector<Scalar> myScalarValues = { {0,255,255},//红色{255,0,255}//紫色 };

      //其中 myColors 和 myScalarValues 的值要一一对应

- Project2_Document_Scanner
    - imgThre=preProcessing(imgOriginal);//先进行图像预处理，得到膨胀后的二值化图片
    - vector<Point> getContours(Mat imgDil)//获取面积最大的矩形的四个点，外部用 vector<Point>initialPoints,docPoints;接收
    - vector<Point> reOrder(vector<Point> points)；docPoints = reOrder(initialPoints);//处理点，按照顺序排好放入 docPoints 中
        - vector<int>sumPoints, subPoints;//分别令每个点的 x+(-)y，并将新的点放入 sumPoints(subPoints)中
        - //按顺序放入 vector<Point>newPoints;中

          //min_element 函数返回的是迭代器，所以要减 sumPoints.begin()/subPoints.begin()，以获得下标

          //为什么如此便是左上点，右上点，左下点，右下点的顺序？画图验证

          newPoints.push_back(points[min_element(sumPoints.begin(), sumPoints.end()) - sumPoints.begin()]);//0

          newPoints.push_back(points[max_element(subPoints.begin(), subPoints.end()) - subPoints.begin()]);//1

          newPoints.push_back(points[min_element(subPoints.begin(), subPoints.end()) - subPoints.begin()]);//2

          newPoints.push_back(points[max_element(sumPoints.begin(), sumPoints.end()) - sumPoints.begin()]);//3

- Mat getWarp(Mat img, vector<Point>docPoints,float w,float h)//透视变换图像处理
    - Mat matrix = getPerspectiveTransform(src, dst);//函数功能：1.根据源图像和目标图像上的四对点坐标来计算从原图像透视变换到目标头像的透视变换矩阵。2.返回变换信息，返回值类型是 Mat
        - src：**源图像中**待测矩形的四点坐标；sdt：**目标图像中**矩形的四点坐标
            - Point2f src[4] = {docPoints[0],docPoints[1],docPoints[2],docPoints[3] };
              Point2f dst[4] = { {0.0f,0.0f},{w,0.0f},{0.0f,h},{w,h} };//浮点数后加 f
- warpPerspective(img, imgWarp, matrix, Size(w, h));函数功能：用于对输入图像进行透视变换

  //输入源图像，目标图像，变换信息，目标图像（窗口）大小
  注：A4 值宽高：float w = 420, h = 596;
    - src：输入图像矩阵

- M：3*3 的透视变换矩阵，可以通过 getPerspectiveTransform 等函数获取
- dsize：结果图像大小，为宽和高的二元组
- dst：输出结果图像，可以省略，结果图像会作为函数处理结果输出

- 裁剪优化 int cropValue = 5;
  Rect roi(cropValue, cropValue, w – 2*cropValue, h –2* cropValue);
  imgCrop = imgWarp(roi);