**基于计算机视觉识别图书条形码**

——084519117 王译

**一、问题重述**

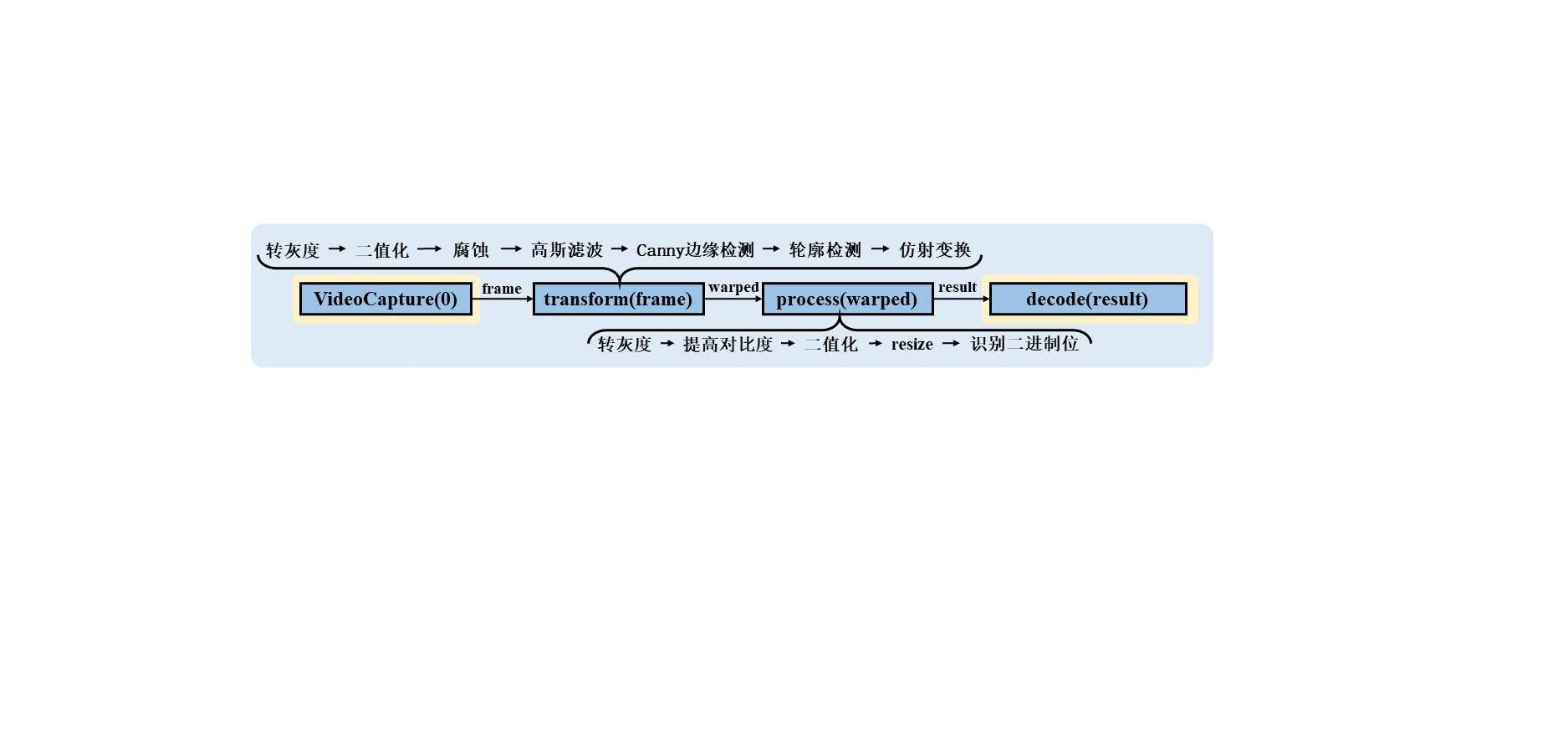
利用计算机视觉相关技术识别出现在摄像头中的图书条形码，并在屏幕上打印识别结果。

**二、问题分析**

条形码是将宽度不等的多个黑条和空白，按照一定的编码规则排列，用以表达一组信息的图形标识符。常见的条形码是由反射率相差很大的黑条（简称条）和白条（简称空）排成的平行线图案。目前的条形码识别依据“条”、“空”对光线具有不同反射率的原理制成，识别效果极佳，而本实验将尝试从计算机视觉领域入手，在像素层次上识别条形码。为了简化问题，降低图像预处理成本，本实验针对图书条形码任务进行。目前关于此方面的参考文献与资源较少。

**三、模型建立**

基于OpenCV库构建条形码识别平台简要pipeline如下：



上述流程图是我经过多次试验得到的效果较好的方案，由于在平常中识别条形码都基于物理性质，真正靠计算机视觉技术来识别条形码的不多，这主要是因为难度大（条形码太细，在对条形码进行操作时，很容易就会影响到条形码本身）和效率低（通过图像处理技术来处理条形码，在有些操作上时间复杂度较高，较难满足实时性的要求）。网上的相关资源也非常少，即使有要么是对电子版条形码进行处理（这样大幅度降低了难度，因为免去了摄像头分辨率以及光照等因素），要么就是直接掉包实现（没法自定义，也没法落地）。

因此我基于多次试验得到了上述的流程，可以实现对笔记本摄像头前的条形码进行实时识别，并且识别错误率控制在1位以内，多数情况下是完全识别正确。

**四、模型求解**

在该部分，我会讲述pipeline中各项的目的与意义，以及在该步的结果，因为pipeline与实际代码流程是有简化的，如果想要完全理解该模型，参见源代码，见附件。

4.1 VideoCapture(0)

目的：调用VideoCapture(0)打开笔记本的内置摄像头，函数返回摄像帧，后续的处理都是基于frame。

效果：



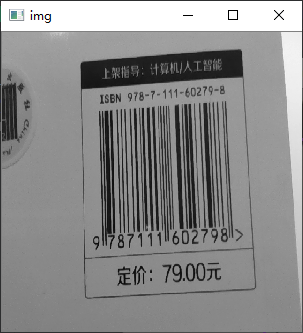
4.2 transform(frame)

该模块里主要实现了提取帧中的条形码区域的功能，最终以仿射变换的条形码图像返回。

4.2.1 转灰度

目的：便于后续的所有操作。

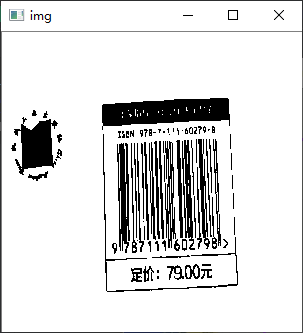
效果：



4.2.2 二值化

目的：这是后续轮廓检测，识别条形码所在区域的必要操作。这里的二值化是简单的阈值二值化，阈值为110，这是一个较为合理的数值，是多次通过直方图确定的。

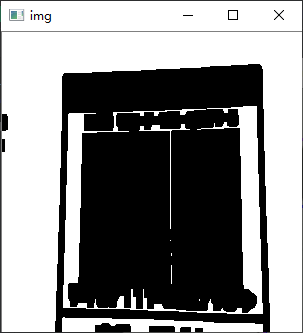
效果：



4.2.3 腐蚀操作

目的：为了后面Canny边缘检测结果的正确，需要腐蚀操作把条形码区域围成一片。腐蚀核大小为5\*5，迭代3次。

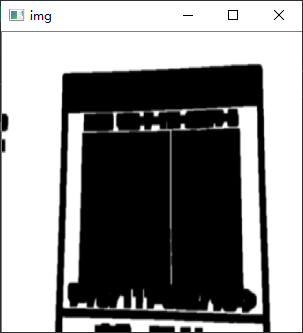
效果：



4.2.4 高斯滤波

目的：上述腐蚀操作之后的图像可能包含噪音点，而且腐蚀操作后的图像边缘也有点锯齿状，采用最常用的高斯滤波来消除噪音和平滑图像。核大小为7\*7，高斯sigma为0。

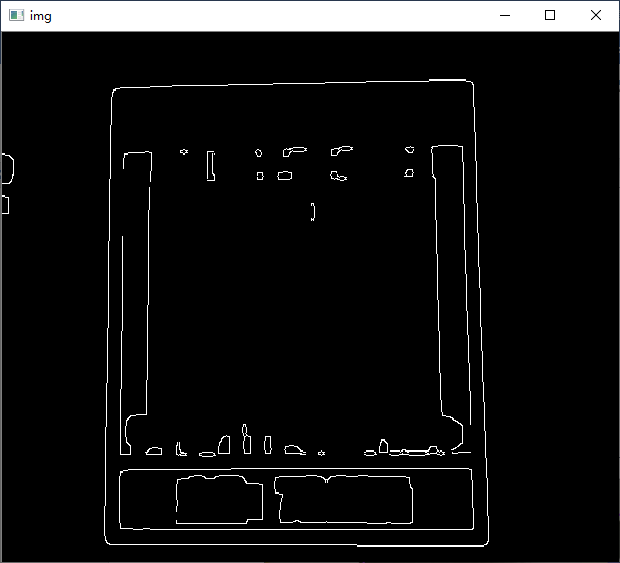
效果：



4.2.5 Canny边缘检测

目的：如果直接对滤波后的图像进行轮廓检测，这样可能导致一些奇怪的轮廓被检测出来，甚至条形码区域都不会被检测出来，但如果通过一次边缘检测，这样最后的轮廓检测效果就比较好。Canny边缘检测的阈值设为75与200.

效果：



4.2.6 轮廓检测

目的：在经过边缘检测后，再进行轮廓检测就可以将条形码区域给画出来了，如下图的蓝框。

效果：



4.2.7 仿射变换

目的：仿射变换的本意是将一组向量放射到另一维度的空间进行表示，在这里的意思就是将条形码区域取出来，以便于后续对图像直接进行处理。仿射变换的原理就是将原图像乘一个仿射变换矩阵M，具体M的计算方法见代码。

效果：



4.3 process(warped)

该模块是对仿射变换后的图像进行操作，返回条形码编码的01列表。

4.3.1 转灰度

目的：在得到上述仿射变化的图像之后，为了便于后续的操作，首先还是必须转灰度图像。

效果：



4.3.2 提高对比度

目的：在转灰度之后，如果直接做二值化分割出条形码的话，效果较差，这是因为条形码与背景图像差不多，都是属于比较灰色的颜色。之前也在这里花费了很长时间，尝试过通过直方图结合OTSU自适应二值化，二值效果都不好。所以为了好做后面的二值化，这里我先提高对比度，把条形码增黑，背景增白。提高对比度函数为自定义实现，原理简要的讲就是设置了fa和fb，在范围内的像素值得到增益：k \* (img[i, j] - fa)，其中k = 255 / (fb - fa)，如果像素值大于fb，直接赋值为255，小于fa，赋值为0。

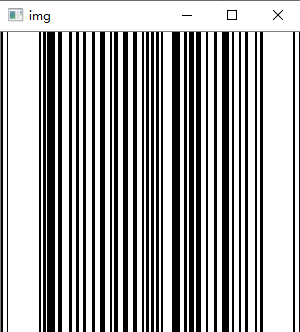
效果：



4.3.3 二值化

目的：在二值化之前其实还有一步，就是img = img2[(imgShape[0] // 2 - 1):(imgShape[0] // 2), :]，即取中间一行的像素值进行处理，这是因为条形码上下看都是一样的，没有必要对整个条形码进行处理，只要取出一行进行识别就好。接着进行二值化，便于后续的识别条形码操作。值得注意的是，这里的参数设置的二值化效果是“非白即黑”（cv2.threshold(img, 254, 255, cv2.THRESH\_BINARY)），这是因为提高对比度之后的图像，有些线是灰色的。

效果：

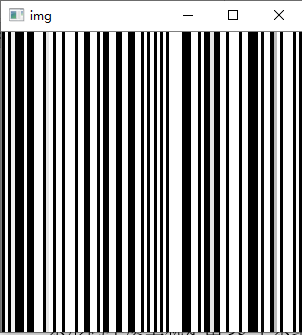


4.3.4 resize

目的：该步之前其实还有一步，就是注意到上面的图，左右边缘都有黑线，这其实是原图包围条形码的框，而不是条形码，所以要在代码里判断黑线是不是条形码，我采用的算法就是算距离，分别从左向右，从右向左扫描，记录遇到的第一个黑线的位置，接着计算位置之差可以得到黑线的相邻距离，太大的肯定是框，直接去除即可，把图像的索引0和索引-1的位置分别置为条形码最左边和最右边的位置。

下面的工作就是识别条形码，为了便于后面分割，先把y长度resize为95，这是因为条形码本质上就是由95个条组成的。

效果：



4.3.5 识别二进制位

目的：检测出条形码的二进制位。到这一步，像素矩阵的维度应该就是1\*95了，那么我们可以依次遍历这95个值，计算他们的像素值白色的占比。我在代码里写的是占比超过30%即认为该条为白色，存储0，否则存储1。30%这个ratio也是经过多次试验得到的，效果较好。

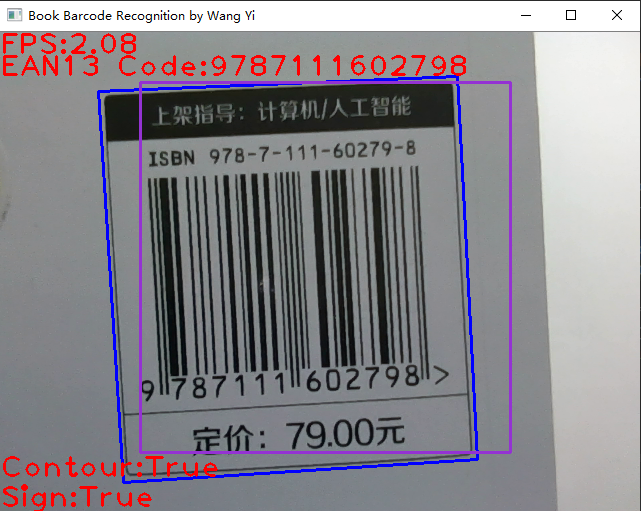
效果：



4.4 decode(result)

目的：根据上述存储二进制位的数组result来解码这些二进制位代表的数字。根据条形码的原理，设置四个字典，分别是左侧奇性字典，左侧偶性字典，右侧偶性字典，首位校验字典，对result进行切割，再在字典中查找，找到一组二进制位对应的十进制数。如果查找失败返回None，该部分详细代码逻辑参见源代码。如果查找成功就将该数字打印到屏幕上，并且替换上一帧解码得到的数字，这样可以确保在经过几帧的运算后，正确的结果得以显现。

效果：



**五、模型的评价与改进方向**

经过我多本书的测试，在绝大多数情况下模型都能正确识别条形码，打印正确的条形码的值，少数情况至多有1位的误差，模型总体的正确率从我直观角度感受还是满意的。并且识别速度也在可以接收的范围内（3s）。**此外为了量化模型结果，找了5本书，每本书做10次检测，共正确48次，正确率为96%。**

未来如果有时间的话，我会进一步优化处理算法，因为现在可以看到在处理条形码时，模型的FPS只有2左右，有点卡卡的。另外可以将条形码识别扩展到二维码识别等领域，其实和条形码大同小异，甚至比条形码简单，因为条形码太细，间隔太小，处理很容易影响条形码。而二维码是由一个个小方块组成的，图像处理不太会影响二维码本身。

**六、我的感想**

大概花了四天时间完成了选题-实践-优化-实现的过程，总体下来收获颇多，因为在暑假期间曾学过OpenCV，所有这次也直接基于OpenCV来做，算是一种复习。除了动手能力，整个课程设计对课内知识也有较大的辅助理解。总的来说，我个人认为该课程设计是一个不错的作品，达到了我的预期。

本学期内课程的学习也是勤奋求索，实验里积极探索MATLAB在图像处理的应用，尽量做到了知其然，知其所以然。

**七、参考文章**

<https://blog.csdn.net/kangkanglhb88008/article/details/86468830>

**八、源代码**

只有一个.py文件，可以直接运行。

|  |
| --- |
| import cv2  import numpy as np  import matplotlib.pyplot as plt  import time  def cv\_show(img, name='img'):  cv2.namedWindow(name, 0)  cv2.resizeWindow(name, 300, 300)  cv2.imshow(name, img)  cv2.waitKey(0)  cv2.destroyAllWindows()  def resize(image, width=None, height=None, inter=cv2.INTER\_AREA):  (h, w) = image.shape[:2]  if width is None and height is None:  return image  if width is None:  r = height / float(h)  dim = (int(w \* r), height)  else:  r = width / float(w)  dim = (width, int(h \* r))  resized = cv2.resize(image, dim, interpolation=inter)  return resized  def order\_points(pts):  # 一共4个坐标点  rect = np.zeros((4, 2), dtype="float32")  # 按顺序找到对应坐标0123分别是 左上，右上，右下，左下  # 计算左上，右下  s = pts.sum(axis=1)  rect[0] = pts[np.argmin(s)]  rect[2] = pts[np.argmax(s)]  # 计算右上和左下  diff = np.diff(pts, axis=1)  rect[1] = pts[np.argmin(diff)]  rect[3] = pts[np.argmax(diff)]  return rect  def four\_point\_transform(image, pts):  # 获取输入坐标点  rect = order\_points(pts)  (tl, tr, br, bl) = rect  # 计算输入的w和h值  widthA = np.sqrt(((br[0] - bl[0]) \*\* 2) + ((br[1] - bl[1]) \*\* 2))  widthB = np.sqrt(((tr[0] - tl[0]) \*\* 2) + ((tr[1] - tl[1]) \*\* 2))  maxWidth = max(int(widthA), int(widthB))  heightA = np.sqrt(((tr[0] - br[0]) \*\* 2) + ((tr[1] - br[1]) \*\* 2))  heightB = np.sqrt(((tl[0] - bl[0]) \*\* 2) + ((tl[1] - bl[1]) \*\* 2))  maxHeight = max(int(heightA), int(heightB))  # 变换后对应坐标位置  dst = np.array([  [0, 0],  [maxWidth - 1, 0],  [maxWidth - 1, maxHeight - 1],  [0, maxHeight - 1]], dtype="float32")  # 计算变换矩阵  M = cv2.getPerspectiveTransform(rect, dst)  warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))  # 返回变换后结果  return warped  def grayContrast(img, fa, fb):  img = img.copy()  k = 255 / (fb - fa)  for i in range(0, img.shape[0]):  for j in range(0, img.shape[1]):  if img[i, j] < fa:  img[i, j] = 0  elif img[i, j] > fb:  img[i, j] = 255  else:  img[i, j] = k \* (img[i, j] - fa)  return img  def transform(image):  orig = image.copy()  image = orig  # 预处理  gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  # ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  ret, thresh = cv2.threshold(gray, 110, 255, cv2.THRESH\_BINARY)  kernal = np.ones((5, 5), np.uint8)  erosion = cv2.erode(thresh, kernal, iterations=3)  blur = cv2.GaussianBlur(erosion, (7, 7), 0)  edged = cv2.Canny(blur, 75, 200)  # 轮廓检测  cnts = cv2.findContours(edged.copy(), cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)[1]  cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]  # 遍历轮廓  for c in cnts:  # 计算轮廓近似  peri = cv2.arcLength(c, True)  # C表示输入的点集  # epsilon表示从原始轮廓到近似轮廓的最大距离，它是一个准确度参数  # True表示封闭的  approx = cv2.approxPolyDP(c, 0.02 \* peri, True)  if peri < 500:  return []  # 4个点的时候就拿出来  if len(approx) == 4:  return approx  return []  def process(warped):  # 转灰度图像  img2 = cv2.cvtColor(warped, cv2.COLOR\_BGR2GRAY)  # 提高对比度  img2 = grayContrast(img2, 50, 100)  # cv\_show(img2)  # 取中间1行像素点进行处理  imgShape = img2.shape  img = img2[(imgShape[0] // 2 - 1):(imgShape[0] // 2), :]  imgShape = img.shape  # 非白即黑，这是因为提高对比度之后的图像，有些线是灰色的  ret, thresh = cv2.threshold(img, 254, 255, cv2.THRESH\_BINARY)  img = thresh  # for i in range(0, imgShape[1]):  # if img[0, i] != 255:  # img[0, i] = 0  # 有些条形码四周有黑框要把他们和条形码区别，算相邻距离即可  blackStartLeft, blackStartRight = [], []  for k in range(1, imgShape[1] // 2): # 从左向右扫描  if img[0, k] == 0 and img[0, k - 1] == 255:  blackStartLeft.append(k)  for k in range(imgShape[1] - 2, imgShape[1] // 2, -1): # 从右向左扫描  if img[0, k] == 0 and img[0, k + 1] == 255:  blackStartRight.append(k)  disBetweenBlackLeft = np.diff(blackStartLeft)  disBetweenBlackRight = np.diff(blackStartRight)  # 结果列表  result = []  if disBetweenBlackLeft != [] and disBetweenBlackRight != []:  if disBetweenBlackLeft[0] - disBetweenBlackLeft[1] > 25:  left = blackStartLeft[1]  else:  left = blackStartLeft[0]  if disBetweenBlackRight[1] - disBetweenBlackRight[0] > 25:  right = blackStartRight[1]  else:  right = blackStartRight[0]  img = img[:, left:right] # 去除黑框，得到条形码结果  # 下面的工作就是识别条形码，为了便于后面分割，先把y长度resize为95  imgShape = img.shape  img = cv2.resize(img, dsize=(95, imgShape[0]))  imgShape = img.shape  # 等分95份  offset = int(imgShape[1] / 95) # 计算偏移量，其实也就是1  for j in range(offset, imgShape[1] + 1):  if j % offset == 0:  ratio = np.sum(img[:, j - offset:j]) / (255 \* offset)  if ratio >= 0.3:  result.append(0)  else:  result.append(1)  # print(result)  if result[:3] == [1, 0, 1] or result[-3:] == [1, 0, 1]: # 条形码的起始符和终止符必须正确  return True, result[3:-3]  else:  return False, result  return False, result  def decode(result):  # 编码表  singularCharDict = {'0001101': 0, '0011001': 1, '0010011': 2,  '0111101': 3, '0100011': 4, '0110001': 5,  '0101111': 6, '0111011': 7, '0110111': 8,  '0001011': 9  }  evenCharDict = {'0100111': 0, '0110011': 1, '0011011': 2,  '0100001': 3, '0011101': 4, '0111001': 5,  '0000101': 6, '0010001': 7, '0001001': 8,  '0010111': 9  }  evenCharDict2 = {'1110010': 0, '1100110': 1, '1101100': 2,  '1000010': 3, '1011100': 4, '1001110': 5,  '1010000': 6, '1000100': 7, '1001000': 8,  '1110100': 9}  firstNumberDict = {'000000': 0, '001011': 1, '001101': 2,  '001110': 3, '010011': 4, '011001': 5,  '011100': 6, '010101': 7, '010110': 8,  '011010': 9}  # 最终的数组与记录奇偶性的列表  number, oddAndEven = [], []  for i in range(7, 42 + 1): # 左侧数据区  if i % 7 == 0:  tempList = list(map(lambda x: str(x), result[i - 7:i])) # 设置为str类型元素的列表  tempStr = ''.join(tempList)  if not singularCharDict.get(tempStr): # 非None值  oddAndEven.append('1')  if not evenCharDict.get(tempStr):  oddAndEven.append('0')  number.append(singularCharDict.get(tempStr) or evenCharDict.get(tempStr))  for i in range(7, 42 + 1): # 右侧数据区  if i % 7 == 0:  tempList = list(map(lambda x: str(x), result[47 + i - 7:47 + i]))  tempStr = ''.join(tempList)  number.append(evenCharDict2.get(tempStr))  number.insert(0, firstNumberDict.get(''.join(oddAndEven))) # 插入奇偶位，也就是第0位  number = ['X' if item == 'None' else item for item in list(map(lambda x: str(x), number))] # 将所有的None值替换为X  return number  def main():  capture = cv2.VideoCapture(0)  number = ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']  if capture.isOpened(): # 检查是否打开正确  openSucess, frame = capture.read()  else:  openSucess = False  while openSucess:  ret, frame = capture.read()  if frame is None:  break  if ret:  # 启动计时器  timeStart = time.time()  # 先进行轮廓检测和投射变换  screenCnt = transform(frame)  ContourDetect = 'False'  SignDetect = 'False'  if screenCnt != []:  ContourDetect = 'True'  cv2.drawContours(frame, [screenCnt], -1, (255, 0, 0), 2) # 绘制轮廓  warped = four\_point\_transform(frame.copy(), screenCnt.reshape(4, 2)) # 投射变换  state, result = process(warped) # 得到投射变换图之后进行图像处理，如果最后条形码起始符和终止符不是101，图片里没有条形码返回false  if state:  numberGenList = decode(result) # decode函数对result里编码进行解码  SignDetect = 'True'  # 替换掉X  number = [numberGenList[i] if item == 'X' and numberGenList[i] != 'X' else item for i, item in  enumerate(number)]  # 替换上一次结果  number = [numberGenList[i] if item != 'X' and numberGenList[i] != 'X' else item for i, item in  enumerate(number)]  # 翻转镜头  # frame = cv2.flip(frame, 180)  timeEND = time.time()  FRAME = 1 / (timeEND - timeStart)  cv2.putText(frame, 'FPS:{:.2f}'.format(FRAME), (0, 22), cv2.FONT\_HERSHEY\_PLAIN, 2.0, (0, 0, 255), 2)  cv2.putText(frame, 'EAN13 Code:' + ''.join(number), (0, 44), cv2.FONT\_HERSHEY\_PLAIN, 2.0, (0, 0, 255), 2)  cv2.putText(frame, 'Contour:' + ContourDetect, (0, 445), cv2.FONT\_HERSHEY\_PLAIN, 2.0, (0, 0, 255), 2)  cv2.putText(frame, 'Sign:' + SignDetect, (0, 475), cv2.FONT\_HERSHEY\_PLAIN, 2.0, (0, 0, 255), 2)  # frameShape = frame.shape  # frame = cv2.resize(frame, (int(frameShape[1] \* 1.4), int(frameShape[0] \* 1.2)))  cv2.rectangle(frame, (140, 420), (510, 50), (211, 50, 148), 2)  cv2.imshow('Book Barcode Recognition by Wang Yi', frame)  if cv2.waitKey(10) & 0xFF == 27:  break  capture.release()  cv2.destroyAllWindows()  main() |