s2-01将车牌抠出来

由于第一阶段裁出来的是彩色图片，需要进用新的参数进行处理。

而取边缘算法改为Canny(做到这时才知有这个算法......想重构，不过己经来不及)，效果相当好。

来自，http://www.kongzhi.net/cases/caseview.php?id=2368 的说法：

Roberts算子检测方法对具有陡峭的低噪声的图像处理效果较好，但是利用roberts算子提取边缘的结果是边缘比较粗，因此边缘的定位不是很准确。

Sobel算子检测方法对灰度渐变和噪声较多的图像处理效果较好，sobel算子对边缘定位不是很准确，图像的边缘不止一个像素。

Prewitt算子检测方法对灰度渐变和噪声较多的图像处理效果较好。但边缘较宽，而且间断点多。

Laplacian算子法对噪声比较敏感，所以很少用该算子检测边缘，而是用来判断边缘像素视为与图像的明区还是暗区。

Canny方法不容易受噪声干扰，能够检测到真正的弱边缘。优点在于，使用两种不同的阈值分别检测强边缘和弱边缘，并且当弱边缘和强边缘相连时，才将弱边缘包含在输出图像中。

|  |
| --- |
| cv::medianBlur(ofo7CharReProcessImg,ofo7CharReProcessImg,11);//中值滤波  cv::cvtColor(ofo7CharReProcessImg,ofo7CharReProcessImg,COLOR\_BGR2GRAY);//转灰度  cv::Canny(ofo7CharReProcessImg,ofo7CharReProcessImg,50,150,3);//取边缘 |



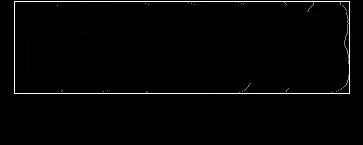
闭操作，用一个长条形将7个数字连起来

|  |
| --- |
| Mat elementOfo7CharReProcess = cv::getStructuringElement(MORPH\_RECT,Size(25,1));//设置膨胀参数  cv::dilate(ofo7CharReProcessImg,ofo7CharReProcessImg,elementOfo7CharReProcess); //膨胀 |



接下来就是外接矩形，并判断是否为车牌

|  |
| --- |
| cv::threshold(ofo7CharReProcessImg,ofo7CharReProcessImg,0,255,CV\_THRESH\_OTSU+CV\_THRESH\_BINARY); //再次二值化，为绘制矩形做准备  vector< vector< Point> > contoursOfo7CharFind; //存放指针  cv::findContours(ofo7CharReProcessImg,contoursOfo7CharFind,CV\_RETR\_EXTERNAL,CV\_CHAIN\_APPROX\_NONE); //找轮廓  for (int i=0;i<contoursOfo7CharFind.size();i++)  {  //绘制最小外接矩形  cv::Rect rectOfo7CharFind=cv::boundingRect(contoursOfo7CharFind[i]);  //判断是否为7个字符  if(  (((float(rectOfo7CharFind.width))/rectOfo7CharFind.height)>3.2)&&  (((float(rectOfo7CharFind.width))/rectOfo7CharFind.height)<4.8)&&  ((rectOfo7CharFind.width>distanceQRx)||(rectOfo7CharFind.width>distanceQRy))  )  {  cv::rectangle(ofo7CharReProcessImg,rectOfo7CharFind,Scalar(255,255,255));  cout<<"Draw7Char: "<< rectOfo7CharFind.width <<"\*"<<rectOfo7CharFind.height<<endl;  Ofo7CharX=rectOfo7CharFind.x;  Ofo7CharY=rectOfo7CharFind.y;  Ofo7CharWidth=rectOfo7CharFind.width;  Ofo7CharHeight=rectOfo7CharFind.height;  Ofo7CharFind++;  }  }  if(Ofo7CharFind!=1)  {  cout<<"ERROR:Can't Find OFO 7 Char position"<<endl;  continue;  } |



s2-02将车牌抠出来

|  |
| --- |
| cv::Rect maskOfo7CharReProcessImg(Ofo7CharX, Ofo7CharY, Ofo7CharWidth,Ofo7CharHeight);  cv::Mat croppedOfo7CharReProcessImg(cropReProcessImg, maskOfo7CharReProcessImg); |



s2-03将7个数字找出来并存放在数组中

|  |
| --- |
| cv::medianBlur(char7findImg,char7findImg,5);//中值滤波  cv::cvtColor(char7findImg,char7findImg,COLOR\_BGR2GRAY);  cv::threshold(char7findImg,char7findImg,0,255,CV\_THRESH\_OTSU+CV\_THRESH\_BINARY); //再次二值化，为绘制矩形做准备  char7findImg.copyTo(char7findImgBw);  vector< vector< Point> > contourschar7findImg; //存放指针  cv::findContours(char7findImg,contourschar7findImg,CV\_RETR\_EXTERNAL,CV\_CHAIN\_APPROX\_NONE); //找轮廓  for (int i=0;i<contourschar7findImg.size();i++)  {  //绘制最小外接矩形  cv::Rect rectRotateReProcess=cv::boundingRect(contourschar7findImg[i]);  if (  (((float(rectRotateReProcess.height))/rectRotateReProcess.width)>1.5)&&  (((float(rectRotateReProcess.height))/rectRotateReProcess.width)<4)&&  (rectRotateReProcess.height>((char7findImg.rows)\*0.67))  )  {  cv::rectangle(char7findImg,rectRotateReProcess,Scalar(255,255,255));  charList[charListCount][1]=rectRotateReProcess.x;  charList[charListCount][2]=rectRotateReProcess.y;  charList[charListCount][3]=rectRotateReProcess.width;  charList[charListCount][4]=rectRotateReProcess.height;  charListCount++;  }  }  if(charListCount!=7)  {  cout<<"ERROR: Can't find ALL 7 char"<<endl;  continue;  } |



s2-04根据x坐标，将数组重排(因为找出来的不是从左到右)

|  |
| --- |
| //冒泡排序  int charSortTemp[5];  for (int i=0;i<7;i++)  {  for (int j=i+1;j<7;j++)  {  if(charList[i][1]>charList[j][1])  {  for(int k=0;k<5;k++)  {  charSortTemp[k]=charList[i][k];  charList[i][k]=charList[j][k];  charList[j][k]=charSortTemp[k];  }  }  }  } |

s2-05将7个数字抠出来，并resize成 32\*72

|  |
| --- |
| for (int i=0;i<7;i++)  {  cout<<filename[i]<<endl;  cv::Rect maskCharList(charList[i][1],charList[i][2],charList[i][3],charList[i][4]);  cv::Mat croppedChar(croppedCharTempImg, maskCharList);  croppedChar.copyTo(croppedCharListImg[i]);  cv::resize(croppedCharListImg[i],croppedCharListImg[i],Size(32,72),0,0,INTER\_CUBIC);  } |

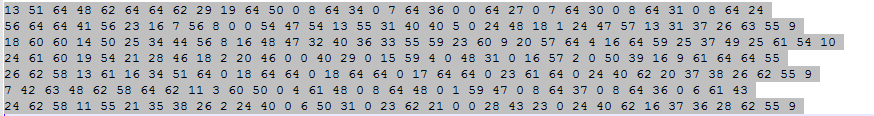


s2-06 将Mat矩阵转换成二维矩阵，因为Mat矩阵只能用一维指针来操作，太麻烦了

|  |
| --- |
| int imgArray[7][72][32];  for(int i=0;i<7;i++)  {  //先二值化  cv::threshold(croppedCharListImg[i],croppedCharListImg[i],0,255,CV\_THRESH\_OTSU+CV\_THRESH\_BINARY);  croppedCharListImg[i].convertTo(croppedCharListImg[i],CV\_8U);  cv::MatIterator\_<cv::Vec3b> it = croppedCharListImg[i].begin<cv::Vec3b>();  int t=0;  for(;it!=croppedCharListImg[i].end<cv::Vec3b>();it++)  {  if((((\*it)[0])==255)&&(((\*it)[1])==255)&&(((\*it)[2])==255))  {  imgArray[i][t/32][t%32]=1;  } else {  imgArray[i][t/32][t%32]=0;  }  t++;  }  } |

s2-07计算feature2(8\*8区域中白点数)

|  |
| --- |
| int feature2Extract[7][32/8\*72/8];  for (int i=0;i<7;i++)  {  int arrayNum=0;  for(int j=0;j<72;j=j+8)  {  for (int k=0;k<32;k=k+8)  {  int countBW=0;  for (int lj=0;lj<8;lj++)  {  for(int lk=0;lk<8;lk++)  {  if ((imgArray[i][j+lj][k+lk])==1)  {  countBW++;  }  }  }  feature2Extract[i][arrayNum]=countBW;  arrayNum++;  }  }  } |



至此图像预处理搞定，可以扔进libsvm做训练了。

SVM相当简单，但要万分注意要CV\_32S

训练

|  |
| --- |
| //转成Mat  Mat trainningDataMat(countLines,32/8\*72/8,CV\_32FC1,trainingData);  Mat labelsMat(countLines,1,CV\_32S,labels); //注意label要用CV\_32S，不能用CV\_32FC1。否则会报错  //训练SVM  Ptr<SVM> svm = SVM::create();  svm->*setType*(SVM::C\_SVC);  svm->*setKernel*(SVM::POLY);  svm->*setDegree*(3);  svm->*setGamma*(0.01);  svm->*setCoef0*(2);  svm->*setTermCriteria*(TermCriteria(TermCriteria::MAX\_ITER, 100, 1e-6));  svm->*train*(trainningDataMat,ROW\_SAMPLE, labelsMat);  svm->*save*("svmtrain.txt"); |

预测(注意预测要逐行逐行来)

|  |
| --- |
| //转成Mat(预测只能逐行逐行来)  for (int i=0;i<countLines;i++)  {  Mat trainningDataMat(1,32/8\*72/8,CV\_32FC1,trainingData[i]);  //预测SVM  cv::Ptr<cv::ml::SVM> mSvm2;  mSvm2 = cv::ml::SVM::load<cv::ml::SVM>("svmtrain.txt");  float q = mSvm2->predict(trainningDataMat);  cout<<"predict: "<<q<<"\treal:"<<labels[i]<<endl;  predictData<<"predict: "<<q<<"\treal:"<<labels[i]<<endl;  } |

成功！Yeah！

