
Fragte am 12.11.2018 >> <<

OpenCV und C++ - Erkennung von Form-und Verkehrszeichen

Ich muss ein Programm schreiben, das 3 Arten von Verkehrszeichen erkennt (Geschwindigkeitsbegrenzung, kein Parken und Warnungen). Ich weiß, wie man einen Kreis mit HoughCircles erkennt, aber ich habe mehrere Bilder und die Parameter für HoughCircles sind für jedes Bild unterschiedlich. Es gibt eine Allgemeine Möglichkeit, Kreise zu erkennen, ohne Parameter für jedes Bild zu ändern?

Außerdem muss ich Dreieck (Warnzeichen) erkennen, also Suche ich nach einem Allgemeinen formdetektor. Haben Sie Vorschläge / code, der mir bei dieser Aufgabe helfen kann?

Schließlich für die Erkennung der Anzahl auf geschwindigkeitsbegrenzungszeichen dachte ich, SIFT verwenden und vergleichen Sie das Bild mit einigen Vorlagen, um die Nummer auf dem Zeichen zu identifizieren. Könnte es ein guter Ansatz sein?

Danke für die Antwort!

c++opencvshapeobject-detection
quelle share

Antwort vom **Gast**

Ich weiß, das ist eine ziemlich alte Frage, aber ich hatte das gleiche problem und jetzt zeige ich Ihnen, wie ich es gelöst habe.
Die folgenden Bilder zeigen einige der genauesten Ergebnisse, die vom opencv-Programm angezeigt werden. In den folgenden Bildern werden die erkannten Straßenschilder mit drei verschiedenen Farben umkreist, die die drei Arten von Straßenschildern unterscheiden (Warnung, kein Parken, Geschwindigkeitsbegrenzung).

Rot für Warnzeichen
Blau für keine parkschilder
Fuchsia für geschwindigkeitsbegrenzungszeichen

Der drehzahlgrenzwert wird grün über den drehzahlgrenzwerten geschrieben

[![example][1]][1]
[![example][2]][2]
[![example][3]][3]
[![example][4]][4]

Wie Sie sehen können, führt das Programm sehr gut, es ist in der Lage zu erkennen und unterscheiden die drei Arten von Zeichen und den geschwindigkeitsgrenzwert im Falle von geschwindigkeitsbegrenzungszeichen erkennen. Alles geschieht, ohne zu viele Fehlalarme zu berechnen, wenn es zum Beispiel im Bild einige Zeichen gibt, die nicht zu einer der drei Kategorien gehören.
Um dieses Ergebnis zu erzielen, berechnet die software die Erkennung in drei Hauptschritten.
Der erste Schritt beinhaltet einen farbbasierten Ansatz, bei dem die roten Objekte im Bild erkannt und Ihre region extrahiert werden, um analysiert zu werden. Dieser Schritt ist besonders nützlich, um die Erkennung von Fehlalarmen zu verhindern, da nur ein kleiner Teil des Bildes verarbeitet wird.
Der zweite Schritt arbeitet mit einem maschinellen Lernalgorithmus: insbesondere verwenden wir einen Kaskadenklassierer, um die Erkennung zu berechnen. Dieser Vorgang erfordert zunächst die Ausbildung der Klassifikatoren und später deren Verwendung zur Erkennung der Zeichen.
Im letzten Schritt werden die Geschwindigkeits-Grenzwerte innerhalb der Geschwindigkeits-Grenzwerte gelesen, auch in diesem Fall durch einen machine learning Algorithmus, jedoch mit dem K-nearest neighbor Algorithmus.
Jetzt werden wir jeden Schritt im detail sehen.

FARBE BASIERTE SCHRITT

Da die Straßenschilder immer von einem roten Rahmen umkreist sind, können wir es uns leisten, nur die Regionen herauszunehmen und zu analysieren, in denen die roten Objekte erkannt werden.
Um die roten Objekte auszuwählen, betrachten wir alle Bereiche der roten Farbe: auch wenn dies einige Fehlalarme erzeugen kann, werden Sie leicht in den nächsten Schritten verworfen werden.

```
inRange(image, Scalar(0, 70, 50), Scalar(10, 255, 255), mask1);
inRange(image, Scalar(170, 70, 50), Scalar(180, 255, 255), mask2);
```

Im Bild unten sehen wir ein Beispiel für die roten Objekte, die mit dieser Methode erkannt wurden.

Nachdem wir die roten Pixel gefunden haben, können wir Sie sammeln, um die Regionen mit einem clustering-Algorithmus zu finden.

```
partition(<#_ForwardIterator __first#>, _ForwardIterator __last, <#_Predicate __pred#>)
```

Nach der Ausführung dieser Methode können wir alle Punkte im selben cluster in einem Vektor speichern (einer für jeden cluster) und die begrenzungsfelder extrahieren, die die
Regionen, die im nächsten Schritt analysiert werden sollen.

HAAR-KASKADEN-KLASSIFIKATOREN FÜR ZEICHENERKENNUNG

Dies ist der eigentliche erkennungsschritt, in dem die Straßenschilder erkannt werden. Um einen kaskadenklassifizierer auszuführen, besteht der erste Schritt darin, einen Datensatz mit positiven und negativen Bildern zu erstellen. Jetzt erkläre ich, wie ich meine eigenen bilderdatensätze erstellt habe.
Das erste, was zu beachten ist, dass wir drei verschiedene haarkaskaden trainieren müssen, um zwischen den drei Arten von Zeichen zu unterscheiden, die wir erkennen müssen, daher müssen wir die folgenden Schritte

für jede der drei Arten von Zeichen wiederholen.

Wir brauchen zwei Datensätze: eine für die positiven Proben (die eine Reihe von Bildern, die die Verkehrszeichen, die wir erkennen, enthält sein müssen) und eine andere für die negativen Proben, die jede Art von Bild ohne Straßenschilder sein kann.
Nachdem wir einen Satz von 100 Bildern für die positiven Proben und einen Satz von 200 Bildern für die negativen in zwei verschiedenen Ordnern gesammelt haben, müssen wir zwei Textdateien schreiben:

Signs.info welche enthält eine Liste von Dateinamen wie die unten,
eine für jede positive Probe im positiven Ordner.

pos/image_name.png 1 0 0 50 45

Hier stellen die zahlen nach dem Namen jeweils die Zahl dar
von Straßenschildern im Bild, die Koordinate der oberen linken
Ecke des straßenschildes, seine Höhe und seine Breite.
Bg.txt welche eine Liste von Dateinamen wie die unten enthält, eine
für jedes Zeichen im negativen Ordner.

neg/street15.png

Mit der Befehlszeile unten erzeugen wir die .vect-Datei, die alle Informationen enthält, die die software aus den positiven Proben abruft.

opencv_createsamples -info sign.info -num 100 -w 50 -h 50 -vec signs.vec

Danach trainieren wir den cascade classifier mit folgendem Befehl:

opencv_traincascade -data data -vec signs.vec -bg bg.txt -numPos 60 -numNeg 200 -numStages 15 -w 50 -h 50 -featureType LBP

wobei die Anzahl der Stufen die Anzahl der Klassifikatoren angibt, die generiert werden, um die Kaskade zu erstellen.
Am Ende dieses Prozesses erhalten wir eine Datei Kaskade.xml, das aus dem Programm CascadeClassifier verwendet wird, um die Objekte im Bild zu erkennen.
Jetzt haben wir unseren Algorithmus trainiert und wir können einen CascadeClassifier für jede Art von straßenschild deklarieren, als wir die Zeichen im Bild erkennen durch

detectMultiScale(<#InputArray image#>, <#std::vector<Rect> &objects#>)

diese Methode erstellt ein Rect um jedes Objekt, das erkannt wurde.
Es ist wichtig zu beachten, dass genau wie jeder machine learning Algorithmus, um gut zu funktionieren, wir eine große Anzahl von Proben in der Datenmenge benötigen. Der Datensatz, den ich erstellt habe, ist nicht extrem groß, daher ist er in einigen Situationen nicht in der Lage, alle Zeichen zu erkennen. Dies geschieht meist, wenn ein kleiner Teil des straßenschildes im Bild nicht sichtbar ist, wie im warnschild unten:

Ich habe meinen Datensatz bis zu dem Punkt erweitert, an dem ich ein ziemlich genaues Ergebnis ohne zu viele Fehler.

ERKENNUNG VON GESCHWINDIGKEITSGRENZWERTEN

Wie für die straßenschildererkennung auch hier habe ich einen maschinellen Lernalgorithmus verwendet, aber mit einem anderen Ansatz. Nach einiger Arbeit erkannte ich, dass eine OCR (tesseract) - Lösung nicht gut funktioniert, also beschloss ich, meine eigene ocr-software zu bauen.

Für den maschinellen Lernalgorithmus nahm ich das Bild unten als Trainingsdaten, die einige geschwindigkeitsgrenzwerte enthält:

Die Anzahl der Trainingsdaten ist gering. Aber da in geschwindigkeitsbegrenzungszeichen alle Buchstaben die gleiche schriftart haben, ist es kein großes problem.
Um die Daten für das training vorzubereiten, habe ich einen kleinen code in OpenCV gemacht. Es macht die folgenden Dinge:

Es lädt das Bild auf der linken Seite;
Es wählt die Ziffern (offensichtlich durch konturfindung und anwenden von Einschränkungen auf Fläche und Höhe der Buchstaben, um Fehlerkennungen zu vermeiden).
Es zeichnet das Begrenzungsrechteck um einen Buchstaben und wartet, bis die Taste manuell gedrückt wird.
Dieses mal drückt der Benutzer die Zifferntaste, die dem Brief im Kasten selbst entspricht.
Sobald die entsprechende Zifferntaste gedrückt wird, speichert Sie 100 Pixelwerte in einem array und die entsprechende manuell eingegebene Ziffer in einem anderen array.
Schließlich speichert es beide arrays in separaten txt-Dateien.

Nach der manuellen stellige Klassifikation alle Ziffern in den zugdaten(Zug.png) sind manuell beschriftet, und das Bild wird wie das folgende Aussehen.

Jetzt gehen wir in die Ausbildung und Prüfung Teil.

Für die Ausbildung machen wir wie folgt:

Laden Sie die txt-Dateien, die wir bereits früher gespeichert
Erstellen Sie eine Instanz von classifier, die wir verwenden werden (KNearest)
Dann verwenden wir Knete.Zug-Funktion, um die Daten zu trainieren

Nun ist die Erkennung:

Wir laden das Bild mit dem geschwindigkeitsbegrenzungszeichen erkannt
Verarbeiten Sie das Bild wie zuvor und extrahieren Sie jede Ziffer mit konturmethoden
Zeichnen Sie dafür einen Begrenzungsrahmen, ändern Sie die Größe auf 10x10 und speichern Sie die Pixelwerte wie zuvor in einem array.
Dann verwenden wir KNearest.find_nearest() Funktion, zum des nächsten Einzelteils zu dem zu finden, das wir Gaben.
Und es erkennt die richtige Ziffer.

Ich habe diese kleine OCR auf vielen Bildern getestet, und gerade mit diesem kleinen Datensatz habe ich eine Genauigkeit von ca.

CODES

Im folgenden poste ich meinen gesamten openCv C++ code in einer einzigen Klasse, nach meiner Anweisung sollten Sie mein Ergebnis erzielen können.

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>
#include <cmath>
#include <stdlib.h>
#include "opencv2/core/core.hpp"
#include "opencv2/highgui.hpp"
#include <string.h>
#include <opencv2/ml/ml.hpp>

using namespace std;
using namespace cv;

std::vector<cv::Rect> getRedObjects(cv::Mat image);
vector<Mat> detectAndDisplaySpeedLimit( Mat frame );
vector<Mat> detectAndDisplayNoParking( Mat frame );
vector<Mat> detectAndDisplayWarning( Mat frame );
void trainDigitClassifier();
string getDigits(Mat image);
vector<Mat> loadAllImage();
int getSpeedLimit(string speed);

//path of the haar cascade files
String no_parking_signs_cascade =
"/Users/giuliopettenuzzo/Desktop/cascade_classifiers/no_parking_cascade.xml";
String speed_signs_cascade =
"/Users/giuliopettenuzzo/Desktop/cascade_classifiers/speed_limit_cascade.xml";
String warning_signs_cascade =
"/Users/giuliopettenuzzo/Desktop/cascade_classifiers/warning_cascade.xml";

CascadeClassifier speed_limit_cascade;
CascadeClassifier no_parking_cascade;
CascadeClassifier warning_cascade;

int main(int argc, char** argv)
{
//train the classifier for digit recognition, this require a manually train, read the report for more details
trainDigitClassifier();

cv::Mat sceneImage;
vector<Mat> allImages = loadAllImage();

for(int i = 0;i<=allImages.size();i++){
sceneImage = allImages[i];

//load the haar cascade files
if( !speed_limit_cascade.load( speed_signs_cascade ) ){ printf("--(!)Error loading\n"); return -1; };
if( !no_parking_cascade.load( no_parking_signs_cascade ) ){ printf("--(!)Error loading\n"); return -1; };
if( !warning_cascade.load( warning_signs_cascade ) ){ printf("--(!)Error loading\n"); return -1; };

Mat scene = sceneImage.clone();

//detect the red objects
std::vector<cv::Rect> allObj = getRedObjects(scene);
```

```
//use the three cascade classifier for each object detected by the getRedObjects() method
for(int j = 0;j<allObj.size();j++){
Mat img = sceneImage(Rect(allObj[j]));
vector<Mat> warningVec = detectAndDisplayWarning(img);
if(warningVec.size()>0){
Rect box = allObj[j];
}
vector<Mat> noParkVec = detectAndDisplayNoParking(img);
if(noParkVec.size()>0){
Rect box = allObj[j];
}
vector<Mat> speedLitmitVec = detectAndDisplaySpeedLimit(img);
if(speedLitmitVec.size()>0){
Rect box = allObj[j];
for(int i = 0; i<speedLitmitVec.size();i++){
//get speed limit and skatch it in the image
int digit = getSpeedLimit(getDigits(speedLitmitVec[i]));
if(digit > 0){
Point point = box.tl();
point.y = point.y + 30;
cv::putText(sceneImage,
"SPEED LIMIT " + to_string(digit),
point,
cv::FONT_HERSHEY_COMPLEX_SMALL,
0.7,
cv::Scalar(0,255,0),
1,
cv::CV__CAP_PROP_LATEST);
}
}
}
}
imshow("currentobj",sceneImage);
waitKey(0);
}
}

/*
* detect the red object in the image given in the param,
* return a vector containing all the Rect of the red objects
*/
std::vector<cv::Rect> getRedObjects(cv::Mat image)
{
Mat3b res = image.clone();
std::vector<cv::Rect> result;

cvtColor(image, image, COLOR_BGR2HSV);

Mat1b mask1, mask2;
//ranges of red color
inRange(image, Scalar(0, 70, 50), Scalar(10, 255, 255), mask1);
inRange(image, Scalar(170, 70, 50), Scalar(180, 255, 255), mask2);

Mat1b mask = mask1 | mask2;
Mat nonZeroCoordinates;
vector<Point> pts;
```

```
findNonZero(mask, pts);
for (int i = 0; i < nonZeroCoordinates.total(); i++ ) {
cout << "Zero#" << i << ": " << nonZeroCoordinates.at<Point>(i).x << ", " <<
nonZeroCoordinates.at<Point>(i).y << endl;
}

int th_distance = 2; // radius tolerance

// Apply partition
// All pixels within the radius tolerance distance will belong to the same class (same label)
vector<int> labels;

// With lambda function (require C++11)
int th2 = th_distance * th_distance;
int n_labels = partition(pts, labels, [th2](const Point& lhs, const Point& rhs) {
return ((lhs.x - rhs.x)*(lhs.x - rhs.x) + (lhs.y - rhs.y)*(lhs.y - rhs.y)) < th2;
});

// You can save all points in the same class in a vector (one for each class), just like findContours
vector<vector<Point>> contours(n_labels);
for (int i = 0; i < pts.size(); ++i){
contours[labels[i]].push_back(pts[i]);
}

// Get bounding boxes
vector<Rect> boxes;
for (int i = 0; i < contours.size(); ++i)
{
Rect box = boundingRect(contours[i]);
if(contours[i].size()>500){//prima era 1000
boxes.push_back(box);

Rect enlarged_box = box + Size(100,100);
enlarged_box -= Point(30,30);

if(enlarged_box.x<0){
enlarged_box.x = 0;
}
if(enlarged_box.y<0){
enlarged_box.y = 0;
}
if(enlarged_box.height + enlarged_box.y > res.rows){
enlarged_box.height = res.rows - enlarged_box.y;
}
if(enlarged_box.width + enlarged_box.x > res.cols){
enlarged_box.width = res.cols - enlarged_box.x;
}

Mat img = res(Rect(enlarged_box));
result.push_back(enlarged_box);
}
}
Rect largest_box = *max_element(boxes.begin(), boxes.end(), [](const Rect& lhs, const Rect& rhs) {
return lhs.area() < rhs.area();
});

//draw the rects in case you want to see them
for(int j=0;j<=boxes.size();j++){
```

```
if(boxes[j].area() > largest_box.area()/3){
rectangle(res, boxes[j], Scalar(0, 0, 255));

Rect enlarged_box = boxes[j] + Size(20,20);
enlarged_box -= Point(10,10);

rectangle(res, enlarged_box, Scalar(0, 255, 0));
}
}

rectangle(res, largest_box, Scalar(0, 0, 255));

Rect enlarged_box = largest_box + Size(20,20);
enlarged_box -= Point(10,10);

rectangle(res, enlarged_box, Scalar(0, 255, 0));

return result;
}

/*
* code for detect the speed limit sign , it draws a circle around the speed limit signs
*/
vector<Mat> detectAndDisplaySpeedLimit( Mat frame )
{
std::vector<Rect> signs;
vector<Mat> result;
Mat frame_gray;

cvtColor( frame, frame_gray, CV_BGR2GRAY );
//normalizes the brightness and increases the contrast of the image
equalizeHist( frame_gray, frame_gray );

//-- Detect signs
speed_limit_cascade.detectMultiScale( frame_gray, signs, 1.1, 3, 0|CV_HAAR_SCALE_IMAGE, Size(30,
30) );
cout << speed_limit_cascade.getFeatureType();

for( size_t i = 0; i < signs.size(); i++ )
{
Point center( signs[i].x + signs[i].width*0.5, signs[i].y + signs[i].height*0.5 );
ellipse( frame, center, Size( signs[i].width*0.5, signs[i].height*0.5), 0, 0, 360, Scalar( 255, 0, 255 ), 4, 8, 0 );


Mat resultImage = frame(Rect(center.x - signs[i].width*0.5,center.y -
signs[i].height*0.5,signs[i].width,signs[i].height));
result.push_back(resultImage);
}
return result;
}

/*
* code for detect the warning sign , it draws a circle around the warning signs
*/
vector<Mat> detectAndDisplayWarning( Mat frame )
{
std::vector<Rect> signs;
vector<Mat> result;
```

```
Mat frame_gray;

cvtColor( frame, frame_gray, CV_BGR2GRAY );
equalizeHist( frame_gray, frame_gray );

//-- Detect signs
warning_cascade.detectMultiScale( frame_gray, signs, 1.1, 3, 0|CV_HAAR_SCALE_IMAGE, Size(30, 30)
);
cout << warning_cascade.getFeatureType();
Rect previus;


for( size_t i = 0; i < signs.size(); i++ )
{
Point center( signs[i].x + signs[i].width*0.5, signs[i].y + signs[i].height*0.5 );
Rect newRect = Rect(center.x - signs[i].width*0.5,center.y -
signs[i].height*0.5,signs[i].width,signs[i].height);
if((previus & newRect).area()>0){
previus = newRect;
}else{
ellipse( frame, center, Size( signs[i].width*0.5, signs[i].height*0.5), 0, 0, 360, Scalar( 0, 0, 255 ), 4, 8, 0 );
Mat resultImage = frame(newRect);
result.push_back(resultImage);
previus = newRect;
}
}
return result;
}

/*
* code for detect the no parking sign , it draws a circle around the no parking signs
*/
vector<Mat> detectAndDisplayNoParking( Mat frame )
{
std::vector<Rect> signs;
vector<Mat> result;
Mat frame_gray;

cvtColor( frame, frame_gray, CV_BGR2GRAY );
equalizeHist( frame_gray, frame_gray );

//-- Detect signs
no_parking_cascade.detectMultiScale( frame_gray, signs, 1.1, 3, 0|CV_HAAR_SCALE_IMAGE, Size(30,
30) );
cout << no_parking_cascade.getFeatureType();
Rect previus;

for( size_t i = 0; i < signs.size(); i++ )
{
Point center( signs[i].x + signs[i].width*0.5, signs[i].y + signs[i].height*0.5 );
Rect newRect = Rect(center.x - signs[i].width*0.5,center.y -
signs[i].height*0.5,signs[i].width,signs[i].height);
if((previus & newRect).area()>0){
previus = newRect;
}else{
ellipse( frame, center, Size( signs[i].width*0.5, signs[i].height*0.5), 0, 0, 360, Scalar( 255, 0, 0 ), 4, 8, 0 );
Mat resultImage = frame(newRect);
result.push_back(resultImage);
```

```
previus = newRect;
}
}
return result;
}

/*
* train the classifier for digit recognition, this could be done only one time, this method save the result in a file and
* it can be used in the next executions
* in order to train user must enter manually the corrisponding digit that the program shows, press space if the red box is just a point (false positive)
*/
void trainDigitClassifier(){
Mat thr,gray,con;
Mat src=imread("/Users/giuliopettenuzzo/Desktop/all_numbers.png",1);
cvtColor(src,gray,CV_BGR2GRAY);
threshold(gray,thr,125,255,THRESH_BINARY_INV); //Threshold to find contour
imshow("ci",thr);
waitKey(0);
thr.copyTo(con);

// Create sample and label data
vector< vector <Point> > contours; // Vector for storing contour
vector< Vec4i > hierarchy;
Mat sample;
Mat response_array;
findContours( con, contours, hierarchy,CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE ); //Find contour

for( int i = 0; i< contours.size(); i=hierarchy[i][0] ) // iterate through first hierarchy level contours
{
Rect r= boundingRect(contours[i]); //Find bounding rect for each contour
rectangle(src,Point(r.x,r.y), Point(r.x+r.width,r.y+r.height), Scalar(0,0,255),2,8,0);
Mat ROI = thr(r); //Crop the image
Mat tmp1, tmp2;
resize(ROI,tmp1, Size(10,10), 0,0,INTER_LINEAR ); //resize to 10X10
tmp1.convertTo(tmp2,CV_32FC1); //convert to float

imshow("src",src);

int c=waitKey(0); // Read corresponding label for contour from keyoard
c-=0x30; // Convert ascii to intiger value
response_array.push_back(c); // Store label to a mat
rectangle(src,Point(r.x,r.y), Point(r.x+r.width,r.y+r.height), Scalar(0,255,0),2,8,0);
sample.push_back(tmp2.reshape(1,1)); // Store sample data
}

// Store the data to file
Mat response,tmp;
tmp=response_array.reshape(1,1); //make continuous
tmp.convertTo(response,CV_32FC1); // Convert to float

FileStorage Data("TrainingData.yml",FileStorage::WRITE); // Store the sample data in a file
Data << "data" << sample;
Data.release();

FileStorage Label("LabelData.yml",FileStorage::WRITE); // Store the label data in a file
```

```
Label << "label" << response;
Label.release();
cout<<"Training and Label data created successfully....!! "<<endl;

imshow("src",src);
waitKey(0);


}

/*
* get digit from the image given in param, using the classifier trained before
*/
string getDigits(Mat image)
{
Mat thr1,gray1,con1;
Mat src1 = image.clone();
cvtColor(src1,gray1,CV_BGR2GRAY);
threshold(gray1,thr1,125,255,THRESH_BINARY_INV); // Threshold to create input
thr1.copyTo(con1);


// Read stored sample and label for training
Mat sample1;
Mat response1,tmp1;
FileStorage Data1("TrainingData.yml",FileStorage::READ); // Read traing data to a Mat
Data1["data"] >> sample1;
Data1.release();

FileStorage Label1("LabelData.yml",FileStorage::READ); // Read label data to a Mat
Label1["label"] >> response1;
Label1.release();


Ptr<ml::KNearest> knn(ml::KNearest::create());

knn->train(sample1, ml::ROW_SAMPLE,response1); // Train with sample and responses
cout<<"Training compleated.....!!"<<endl;

vector< vector <Point> > contours1; // Vector for storing contour
vector< Vec4i > hierarchy1;

//Create input sample by contour finding and cropping
findContours( con1, contours1, hierarchy1,CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE );
Mat dst1(src1.rows,src1.cols,CV_8UC3,Scalar::all(0));
string result;

for( int i = 0; i< contours1.size(); i=hierarchy1[i][0] ) // iterate through each contour for first hierarchy level .
{
Rect r= boundingRect(contours1[i]);
Mat ROI = thr1(r);
Mat tmp1, tmp2;
resize(ROI,tmp1, Size(10,10), 0,0,INTER_LINEAR );
tmp1.convertTo(tmp2,CV_32FC1);
Mat bestLabels;
float p=knn -> findNearest(tmp2.reshape(1,1),4, bestLabels);
char name[4];
sprintf(name,"\%d",(int)p);
```

```
cout << "num = " << (int)p;
result = result + to_string((int)p);

putText( dst1,name,Point(r.x,r.y+r.height) ,0,1, Scalar(0, 255, 0), 2, 8 );
}

imwrite("dest.jpg",dst1);
return result ;
}
/*
* from the digits detected, it returns a speed limit if it is detected correctly, -1 otherwise
*/
int getSpeedLimit(string numbers){
if ((numbers.find("30") != std::string::npos) || (numbers.find("03") != std::string::npos)) {
return 30;
}
if ((numbers.find("50") != std::string::npos) || (numbers.find("05") != std::string::npos)) {
return 50;
}
if ((numbers.find("80") != std::string::npos) || (numbers.find("08") != std::string::npos)) {
return 80;
}
if ((numbers.find("70") != std::string::npos) || (numbers.find("07") != std::string::npos)) {
return 70;
}
if ((numbers.find("90") != std::string::npos) || (numbers.find("09") != std::string::npos)) {
return 90;
}
if ((numbers.find("100") != std::string::npos) || (numbers.find("001") != std::string::npos)) {
return 100;
}
if ((numbers.find("130") != std::string::npos) || (numbers.find("031") != std::string::npos)) {
return 130;
}
return -1;
}

/*
* load all the image in the file with the path hard coded below
*/
vector<Mat> loadAllImage(){
vector<cv::String> fn;
glob("/Users/giuliopettenuzzo/Desktop/T1/dataset/*.jpg", fn, false);

vector<Mat> images;
size_t count = fn.size(); //number of png files in images folder
for (size_t i=0; i<count; i++)
images.push_back(imread(fn[i]));
return images;
}
```

Bewertung
Keine Antwort gefunden?

Nutzen Sie die Suche in der gesamten Datenbank der Website und finden Sie die Antwort für sich.

Weitere Antworten finden
Auswählen

Bild hochladen
Verwandte Probleme
1

antwort

Fragte am 20.11.2018

GLEW richtig auf Einem Projekt verknüpft, aber nicht ein Anderes (C++ mit MinGW Make)

Ich muss hier etwas vermissen, und das ist zu erwarten, da ich vor kurzem C++ aufgegriffen habe. Aber ich habe ein Projekt mit MinGW Make kompiliert, und es kommt immer mit fehlenden Referenzen für...

1

antwort

Fragte am 20.11.2018

Werden Basisklassen in der Vererbung in die abgeleitete Klasse kopiert?

Wie der Titel sagt: werden Basisklassen in der Vererbung in die abgeleitete Klasse kopiert, meine ich wie #include der code aus der Basisklasse wird in die abgeleitete Klasse kopiert...

1

antwort

Fragte am 20.11.2018

Sort_index () Funktion in Gürteltier Bibliothek gibt Falsches Ergebnis

Ich versuche, das zu verwenden sort_index() Funktion in der Armadillo C++ - Bibliothek (link hier). Hier...

1

antwort

Fragte am 20.11.2018

Benötigen Sie Hilfe zum OOP-design für das teilen von Variablen zwischen Klassen, die mit einem timer unabhängig laufen

Nehmen wir an, ich habe 2 Klassen Fly und Bee, welche Methoden (bzw. /* Fly */ using namespace boost::asio using namespace boost::posix_time class Fly { ... deadline_timer...

1

antwort

Fragte am 20.11.2018

Kann RAND_event () im message-handler zu wenig freezes in der UI führen?

Es gibt einen Meldungshandler (in C++ Builder) wie folgt: void __fastcall TMainForm::HandleMessages(tagMSG &Msg, bool &Handled) { RAND_event(Msg.message, Msg.wParam, Msg.lParam); ...

Stelle eine Frage

- DevTimes.de - 2018
  Entwicklergemeinschaft