ООО «Ангелы АйТи»
ОГРН 1103668001227
ИНН/КПП 3664101629/366601001
394036, г.Воронеж
ул. Карла Маркса, д.70 А, офис 505
ОКПО 63862783
Р/с 40702810813000003671
в Центрально-Черноземном банке
Сбербанка РФ г.Воронеж
БИК 042007681
К/с 30101810600000000681
Тел. +7(473) 2-555-007
Факс +7(473) 2-555-655

Библиотека разработчика SDK «Angel:Vision».

SDK «Angel:Vision» предназначена для автоматической идентификации номеров транспортных средств, попавших в поле зрения видеокамеры. Модуль распознает все шаблоны номерных знаков России с высоким показателем до 99%. Фиксация регистрационных знаков транспортных средств может осуществляться одновременно для неограниченного количества автомобилей, попавших в кадр, на нескольких полосах движения, включая противоположно направленный трафик.

Описание SDK:

Класс Options

Настройки распознавания государственных регистрационных знаков автотранспорта.

Пространство имен: Angel.Vision.LicensePlate
Сборка: Angel.Vision.dll

Синтаксис

C++

```
LPR_Options
```

Конструкторы

Имя	Описание
`LPR_Options()`	Инициализирует новый экземпляр класса LPR_Options значениями по умолчанию.

Свойства

Имя	Описание
`bool Tracking`	Включает/выключает динамический режим обработки кадров
`int NumberFramesForLose`	Количество кадров без обнаружения номера для того что бы посчитать его потерянным (установить флаг Lost в результате)
`int MinNumberFramesForTrack`	Минимальное количество кадров с обнаруженным номером что бы начать слежение (если кадров меньше чем установлено то флаг Lost не устанавливается да же при потере)
`int MaxPlateWidth`	Максимальная ширина номерной пластины.
`int MinPlateWidth`	Минимальная ширина номерной пластины.
`double Accuracy`	Точность поиска номерной пластины, нижняя граница значения 1.01 сверху не ограниченно. Чем меньше значение тем точнее и лучше ищет номер в кадре но больше времени затрачивается на обработку одного кадра.
`int Group`	Группа обрабатываемых номерных пластин используется для фильтра ложных срабатываний, нижняя граница 0 сверху не ограниченно. Чем больше значение тем меньше ложных срабатываний.
`bool SelectionByUndefined`	Включает\выключает отбор результатов по количеству

	нераспознанных символов.
int MaxCountUndefinedSymbol	Максимальное количество нераспознанных символов.
bool SelectionByProbability	Включает\выключает отбор результатов по вероятности распознавания.
double MinProbability	Минимальная вероятность распознавания, диапазон значений от 0 до 1.
int MaxPlatesCount	Ограничивает количество номерных знаков которые ищутся в кадре, если значение 0 то ограничение не устанавливается.
int TotalSecondsForDuplicates	Количество секунд, которое информация о зафиксированной номерной пластине хранится в памяти что бы, если номерная пластина вновь будет найдена, при ее повторной потери выдавать статус повторной фиксации.

Класс LicensePlate

Результат распознавания регистрационных номеров автотранспорта.

Пространство имен: Angel.Vision.LicensePlate
Сборка: Angel.Vision.dll

Синтаксис

C++

```
LicensePlate
```

Свойства

Имя	Описание
Rect Position	Прямоугольник указывающий на позицию номерной пластины в кадре.
vector<Point> Points	Координаты точек краев номерной пластины, с учетом определенного угла наклона.
string Number	Распознанные номер.
double Probability	Вероятность распознавания.
int Status	Статус слежения за номерной пластиной 0 номерная пластина находится в зоне видимости, 1 – номерная пластина потеряна, 2 – номерная пластина потеряна повторна, происходит когда по каким то причинам номер был уже потерян, а через короткое время, заданное в настройках, был снова обнаружен.
int Index	Индекс номерной пластины в истории зафиксированных номерных пластин.
Mat BestImage	Лучший кадр. Только для динамического режима.
time_t Time	Дата и время лучшего кадра.

Класс LicensePlateRecognizer

Реализует методы распознавания регистрационных номеров транспортных средств используя данные о пикселях графического изображения.

Пространство имен: Angel.Vision.LicensePlate
Сборка: Angel.Vision.dll

Синтаксис

C++

```
LicensePlateRecognizer
```

Конструкторы

Имя	Описание
LicensePlateRecognizer()	Инициализирует новый экземпляр класса LicensePlateRecognizer

Методы

Имя	Описание
void SetOptions(LPR_Options& opt)	Устанавливает настройки модуля распознавания
void AddArea(Rect& area)	Добавляет зону поиска номерной пластины в кадре.
void ClearAreas()	Очищает зоны поиска номерной пластины в кадре.
int GetAreasCount()	Возвращает количество установленных сон поиска номерной пластины в кадре.
void Recognize(Mat& img, vector<LicensePlate>& plates, time_t t = 0)	Выполняет распознавания изображения и возвращает результат распознавания, с установкой временной отметки (имеет смысл только для динамического режима).

Примеры

Следующий пример демонстрирует распознавание номеров по видео файлу.

```
#include "Recognizers.h"
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
#include <clocale>
#include <map>

using namespace std;
using namespace cv;
using namespace Angel::Vision::LicensePlate;


int main(int argc, char* argv[])
{
	setlocale(LC_CTYPE, "rus");

	VideoCapture capture;
	Mat frame;
	Mat smal;

	LicensePlateRecognizer lpr = LicensePlateRecognizer();

	LPR_Options opt = LPR_Options();

	opt.MinPlateWidth = 80;
	opt.MaxPlateWidth = 240;
	opt.Accuracy = 1.5;
	opt.Tracking = true;
	opt.MinNumberFramesForTrack = 1;
	opt.NumberFramesForLose = 5;
	opt.SelectionByUndefined = false;
	opt.MaxCountUndefinedSymbol = 0;
	opt.SelectionByProbability = false;
	opt.MinProbability = 0.0;

	string fileName = "";
	string directory = "";

	for (int i = 1; i < argc; i++)
	{
		bool set = false;
		if (!strcmp(argv[i], "-minPlateWidth"))
```

```
		{
			opt.MinPlateWidth = atoi(argv[++i]);
		}
		else if (!strcmp(argv[i], "-maxPlateWidth"))
		{
			opt.MaxPlateWidth = atoi(argv[++i]);
		}
		else if (!strcmp(argv[i], "-accuracy"))
		{
			opt.Accuracy = atof(argv[++i]);
		}
		else if (!strcmp(argv[i], "-angle"))
		{
			opt.Angle = atof(argv[++i]);
		}
		else if (!strcmp(argv[i], "-autoRotate"))
		{
			opt.AutoRotate = atoi(argv[++i]) > 0;
		}
		else if (!strcmp(argv[i], "-tracking"))
		{
			opt.Tracking = atoi(argv[++i]);
		}
		else if (!strcmp(argv[i], "-group"))
		{
			opt.Group = atoi(argv[++i]);
		}
		else if (!strcmp(argv[i], "-minProbability"))
		{
			opt.SelectionByProbability = true;
			opt.MinProbability = atof(argv[++i]);
		}
		else if (!strcmp(argv[i], "-limitSimilarity"))
		{
			opt.LimitSimilarity = atof(argv[++i]);
		}
		else if (!strcmp(argv[i], "-maxCountUndefinedSymbol"))
		{
			opt.SelectionByUndefined = true;
			opt.MaxCountUndefinedSymbol = atoi(argv[++i]);
		}
		else if (!strcmp(argv[i], "-fileName"))
		{
			fileName = string(argv[++i]);
		}
		else if (!strcmp(argv[i], "-directory"))
		{
			directory = string(argv[++i]);
		}

	}

	if (!capture.open(fileName))
		cout << "Erro create " << fileName << " capture." << endl;

	int width = capture.get(CV_CAP_PROP_FRAME_WIDTH);
	int height = capture.get(CV_CAP_PROP_FRAME_HEIGHT);

	lpr.SetOptions(opt);

	lpr.StartBackgraund();

	int count = 0;

	time_t ct1;

	time(&ct1);
	unsigned int id = 1;

	map<string, string> cars = map<string, string>();

	while (capture.read(frame))
	{
```

```
count++;

Mat frame1 = frame.clone();

int b = 0;

vector<LicensePlate> plates;

time_t t;
time(&t);

lpr.Recognize(frame1, plates, id, t);

id++;

for (size_t i = 0; i < plates.size(); i++)
{
	b = 1;

	Mat ff1;

	if (plates[i].Status == LPR_LOST)
	{
		ff1 = plates[i].BestImage;

		line(ff1, cvPoint(plates[i].Points[0].x, plates[i].Points[0].y),
			cvPoint(plates[i].Points[1].x, plates[i].Points[1].y),
			Scalar(255, 0, 0), 3, 8, 0);
		line(ff1, cvPoint(plates[i].Points[1].x, plates[i].Points[1].y),
			cvPoint(plates[i].Points[2].x, plates[i].Points[2].y),
			Scalar(255, 0, 0), 3, 8, 0);
		line(ff1, cvPoint(plates[i].Points[2].x, plates[i].Points[2].y),
			cvPoint(plates[i].Points[3].x, plates[i].Points[3].y),
			Scalar(255, 0, 0), 3, 8, 0);
		line(ff1, cvPoint(plates[i].Points[3].x, plates[i].Points[3].y),
			cvPoint(plates[i].Points[0].x, plates[i].Points[0].y),
			Scalar(255, 0, 0), 3, 8, 0);
	}
	else if (plates[i].Index == -1)
	{
		line(frame, cvPoint(plates[i].Points[0].x, plates[i].Points[0].y),
			cvPoint(plates[i].Points[1].x, plates[i].Points[1].y),
			Scalar(255, 0, 0), 3, 8, 0);
		line(frame, cvPoint(plates[i].Points[1].x, plates[i].Points[1].y),
			cvPoint(plates[i].Points[2].x, plates[i].Points[2].y),
			Scalar(255, 0, 0), 3, 8, 0);
		line(frame, cvPoint(plates[i].Points[2].x, plates[i].Points[2].y),
			cvPoint(plates[i].Points[3].x, plates[i].Points[3].y),
			Scalar(255, 0, 0), 3, 8, 0);
		line(frame, cvPoint(plates[i].Points[3].x, plates[i].Points[3].y),
			cvPoint(plates[i].Points[0].x, plates[i].Points[0].y),
			Scalar(255, 0, 0), 3, 8, 0);
	}
	else if (plates[i].Index == -2)
	{
		line(frame, cvPoint(plates[i].Points[0].x, plates[i].Points[0].y),
			cvPoint(plates[i].Points[1].x, plates[i].Points[1].y),
			Scalar(0, 255, 0), 3, 8, 0);
		line(frame, cvPoint(plates[i].Points[1].x, plates[i].Points[1].y),
			cvPoint(plates[i].Points[2].x, plates[i].Points[2].y),
			Scalar(0, 255, 0), 3, 8, 0);
		line(frame, cvPoint(plates[i].Points[2].x, plates[i].Points[2].y),
			cvPoint(plates[i].Points[3].x, plates[i].Points[3].y),
			Scalar(0, 255, 0), 3, 8, 0);
		line(frame, cvPoint(plates[i].Points[3].x, plates[i].Points[3].y),
			cvPoint(plates[i].Points[0].x, plates[i].Points[0].y),
			Scalar(0, 255, 0), 3, 8, 0);
	}
	else
	{
		line(frame, cvPoint(plates[i].Points[0].x, plates[i].Points[0].y),
			cvPoint(plates[i].Points[1].x, plates[i].Points[1].y),
			Scalar(0, 0, 255), 3, 8, 0);
```

```
                        line(frame, cvPoint(plates[i].Points[1].x, plates[i].Points[1].y),
                                cvPoint(plates[i].Points[2].x, plates[i].Points[2].y),
                                Scalar(0, 0, 255), 3, 8, 0);
                        line(frame, cvPoint(plates[i].Points[2].x, plates[i].Points[2].y),
                                cvPoint(plates[i].Points[3].x, plates[i].Points[3].y),
                                Scalar(0, 0, 255), 3, 8, 0);
                        line(frame, cvPoint(plates[i].Points[3].x, plates[i].Points[3].y),
                                cvPoint(plates[i].Points[0].x, plates[i].Points[0].y),
                                Scalar(0, 0, 255), 3, 8, 0);
                }

                std::stringstream ss;
                ss << plates[i].Index;

                cout << plates[i].Number<<" - "<< plates[i].Index << " - " << plates[i].Time
<< " - " << plates[i].Probability << ":";

                map<string, string>::iterator car = cars.find(plates[i].Number);


                if (plates[i].Status == LPR_LOST)
                {
                        string number = plates[i].Number;

                        for (int ii = 0; ii < number.length(); ++ii)
                        {
                                if (number[ii] == '*')
                                        number[ii] = '_';
                        }

                        imwrite(directory + "\\" + ss.str()  + "_" + number + ".jpg", ff1);
                }
        }

        if (plates.size() > 0)
                cout << endl;

        plates.clear();

        if (count % 5 == 0)
        {
                cv::resize(frame, smal, Size(), 0.5, 0.5, INTER_LINEAR);

                cv::imshow("result", smal);
        }

        char c = (char)waitKey(1);
        if (c == 27 || c == 'q' || c == 'Q')
                break;
    }

    return 0;
}
```