#pragma once

// dimNoteDimRecog.h - Contains declaration of Function class

#pragma once

#ifdef dimNoteDimRecog\_EXPORTS

#define dimNoteDimRecog\_API \_\_declspec(dllexport)

#else

#define dimNoteDimRecog\_API \_\_declspec(dllimport)

#endif

#include<opencv2/core/core.hpp>

#include<opencv2/highgui/highgui.hpp>

#include<opencv2/imgproc/imgproc.hpp>

#include<iostream>

#include<conio.h> // may have to modify this line if not using Windows

#include <iostream>

#include <opencv2/opencv.hpp>

#include <algorithm>

#include <math.h>

#include<opencv2/ml/ml.hpp>

#include <string>

#include<sstream>

#include <fstream>

#include <Tchar.h>

#include <io.h>

#include <fcntl.h>

#include <locale>

#include <codecvt>

#ifdef \_MSC\_VER

#define \_CRT\_SECURE\_NO\_WARNINGS

#endif

using namespace cv;

using namespace std;

namespace dimNoteDimRecog

{

// This class is exported from the dimNoteDimRecog.dll

// global constants ///////////////////////////////////////////////////////////////////////////////

const cv::Scalar SCALAR\_BLACK = cv::Scalar(0.0, 0.0, 0.0);

const cv::Scalar SCALAR\_WHITE = cv::Scalar(255.0, 255.0, 255.0);

const cv::Scalar SCALAR\_YELLOW = cv::Scalar(0.0, 255.0, 255.0);

const cv::Scalar SCALAR\_GREEN = cv::Scalar(0.0, 255.0, 0.0);

const cv::Scalar SCALAR\_BLUE = cv::Scalar(255.0, 0.0, 0.0);

const cv::Scalar SCALAR\_RED = cv::Scalar(0.0, 0.0, 255.0);

const double dimGDTNoteSet\_WIDTH\_PADDING\_FACTOR = 1.3;

const double dimGDTNoteSet\_HEIGHT\_PADDING\_FACTOR = 1.5;

const int MIN\_PIXEL\_WIDTH = 2;

const int MIN\_PIXEL\_HEIGHT = 8;

const double MIN\_ASPECT\_RATIO = 0.1;

const double MAX\_ASPECT\_RATIO = 0.9;

const int MIN\_PIXEL\_AREA = 100;

// constants for comparing two chars

const double MIN\_DIAG\_SIZE\_MULTIPLE\_AWAY = 0.01;

const double MAX\_DIAG\_SIZE\_MULTIPLE\_AWAY = 5.0;

const double MAX\_CHANGE\_IN\_AREA = 0.25;

const double MAX\_AREA = 2000;

const double MAX\_CHANGE\_IN\_WIDTH = 0.8;

const double MAX\_CHANGE\_IN\_HEIGHT = 0.2;

// other constants

const int MIN\_NUMBER\_OF\_MATCHING\_CHARS = 2;

const int RESIZED\_CHAR\_IMAGE\_WIDTH = 20;

const int RESIZED\_CHAR\_IMAGE\_HEIGHT = 30;

const int MIN\_CONTOUR\_AREA = 150;

// global variables ///////////////////////////////////////////////////////////////////////////////

const cv::Size GAUSSIAN\_SMOOTH\_FILTER\_SIZE = cv::Size(5, 5);

const int ADAPTIVE\_THRESH\_BLOCK\_SIZE = 19;

const int ADAPTIVE\_THRESH\_WEIGHT = 9;

// external global variables //////////////////////////////////////////////////////////////////////

extern const bool blnShowSteps;

extern cv::Ptr<cv::ml::KNearest> kNearest;

///////////////////////////////////////////////////////////////////////////////////////////////////

class DimSet {

public:

// member variables ///////////////////////////////////////////////////////////////////////////

cv::Mat imageDimSet;

cv::Mat imageDimSetGSBW;

cv::Mat imageDimSetThreshold;

cv::RotatedRect dimSetLocations;

std::string dimNoteStrings;

std::string accuracyPercentage;

std::string confidencePercentage;

///////////////////////////////////////////////////////////////////////////////////////////////

static bool sortDescendingByNumberOfChars(const DimSet &ppLeft, const DimSet &ppRight) {

return(ppLeft.dimNoteStrings.length() > ppRight.dimNoteStrings.length());

}

};

///////////////////////////////////////////////////////////////////////////////////////////////////

class opVector {

public:

std::vector<DimSet> vectorOfdimGDTNoteSet1;

std::vector<DimSet> vectorOfdimGDTNoteSet2;

std::vector<DimSet> vectorOfdimGDTNoteSet3;

std::vector<DimSet> vectorOfdimGDTNoteSet4;

};

///////////////////////////////////////////////////////////////////////////////////////////////////

class Character {

public:

// member variables ///////////////////////////////////////////////////////////////////////////

std::vector<cv::Point> contour;

cv::Rect boundingRect;

int intCenterX;

int intCenterY;

double dblDiagonalSize;

double dblAspectRatio;

///////////////////////////////////////////////////////////////////////////////////////////////

static bool sortCharsLeftToRight(const Character &pcLeft, const Character & pcRight) {

return(pcLeft.intCenterX < pcRight.intCenterX);

}

///////////////////////////////////////////////////////////////////////////////////////////////

bool operator == (const Character& otherCharacter) const {

if (this->contour == otherCharacter.contour) return true;

else return false;

}

///////////////////////////////////////////////////////////////////////////////////////////////

bool operator != (const Character& otherCharacter) const {

if (this->contour != otherCharacter.contour) return true;

else return false;

}

// function prototypes ////////////////////////////////////////////////////////////////////////

Character(std::vector<cv::Point> \_contour);

};

//class Functions

//{

// function prototypes ////////////////////////////////////////////////////////////////////////////

std::vector<DimSet> dimNoteDimRecognition(char\* ipFilename, int knn, int horVerMode);

void drawRectAroundDimSet(cv::Mat &inputImage, DimSet &dimensionSet);

std::vector<DimSet> detectDimSet(cv::Mat &inputImage);

std::vector<Character> findCharactersFromInput(cv::Mat &imageDimSetThreshold);

DimSet extractDimSet(cv::Mat &imgOriginal, std::vector<Character> &vectorOfMatchingChars);

bool loadKNNDataAndTrainKNN(void);

std::vector<DimSet> detectChars(std::vector<DimSet> &vectorOfdimGDTNoteSet);

std::vector<Character> findChars(cv::Mat &imageDimSetGSBW, cv::Mat &imageDimSetThreshold);

bool checkIfCharacter(Character &Character);

std::vector<std::vector<Character> > findVectorOfVectorsOfMatchingChars(const std::vector<Character> &vectorOfCharacters);

std::vector<Character> findVectorOfMatchingChars(const Character &charac, const std::vector<Character> &vectorOfChars);

double distanceBetweenChars(const Character &firstChar, const Character &secondChar);

double angleBetweenChars(const Character &firstChar, const Character &secondChar);

std::vector<Character> removeInnerOverlappingChars(std::vector<Character> &vectorOfMatchingChars);

std::vector<std::string> recognizeCharsInDimSet(cv::Mat &imageDimSetThreshold, std::vector<Character> &vectorOfMatchingChars);

//};

}