

**Mestrado Integrado em Engenharia Informática e Computação**

**5º Ano**

**1º Semestre**

**Realidade Aumentada com Marcas Naturais**

**Realidade Virtual e Aumentada**

**2017/2018**

**Paulo Jorge Silva Ferreira,** [**up201305617@fe.up.pt**](mailto:up201305617@fe.up.pt)

**20 de dezembro de 2017**

Índice

[Introdução 3](#_Toc501635092)

[Programa de Preparação 3](#_Toc501635093)

[Programa de Aumento 4](#_Toc501635094)

[Resultados 7](#_Toc501635095)

[Conclusão 8](#_Toc501635096)

[Bibliografia 8](#_Toc501635097)

[Anexos 9](#_Toc501635098)

[A: Código 9](#_Toc501635099)

[Augmentation.h 9](#_Toc501635100)

[Preparation.h 10](#_Toc501635101)

[Preparation.cpp 11](#_Toc501635102)

[Augmentation.cpp 13](#_Toc501635103)

[Main.cpp 19](#_Toc501635104)

## Introdução

No âmbito da unidade curricular Realidade Virtual e Aumentada foi proposto o desenvolvimento de um conjunto de programas que possam ser usados para aumentar imagens de edifícios. O projeto está divido em dois programas: o de Preparação e o de Aumento.

O programa de Preparação permite a criação de uma base de dados com imagens dos edifícios que se pretende aumentar. Para isso o utilizador define uma linha delimitadora da região da imagem que pretende para aumentar outras imagens do edifício.

O programa de Aumento permite aumentar uma imagem do edifico numa pose diferente da imagem usada no programa de preparação. O programa mostra as marcas associadas ao edifício nas posições corretas, senão for possível deve notificar o utilizador.

## Programa de Preparação

No enunciado era pedido que o utilizador pudesse associar às imagens marcas, tais como linhas delimitadoras, setas ou etiquetas, mas neste programa optou-se apenas por implementar as linhas delimitadoras sob a forma de retângulos.



*Fig. 1 – Retângulo delimitador a azul*

O programa permite abrir uma imagem e selecionar um ou mais retângulos delimitadores através da função **void cv::selectROIs (const String &windowName, InputArray img, std::vector< Rect > &boundingBoxes, bool showCrosshair=true, bool fromCenter=false)**, da biblioteca OpenCV. Esta função cria uma janela com a imagem e permite ao utilizador selecionar as regiões de interesse (ROI) utilizando o rato. Utilizar a tecla ENTER para terminar a seleção da atual região e começar uma nova. Para terminara a seleção de regiões de interesse utilizar a tecla ESC.

Depois de terminar a seleção de ROI’s, a janela onde a imagem está a ser exibida é fechada e na consola o utilizador deve atribuir nomes as regiões que selecionou. Posteriormente as ROI’s são guardadas na base de dados com os nomes que o utilizador atribui.

O utilizador pode agora usar outra imagem para extrair as regiões de interesse ou então avançar para o programa de Aumento.

## Programa de Aumento

Inicialmente o programa de aumento permite ao utilizador escolher o Feature Detector (FAST, SIFT, SURF, ORB), o Descriptor Extractor (SIFT, SURF, ORB, BRIEF, FREAK) e o Descriptor Matcher (BFMatcher, FlannBasedMatcher). Por fim escolhe a imagem que pretende aumentar.

O programa começa por detetar os keypoints da imagem que pretende aumentar e extrair os descritores desses mesmos keypoints.



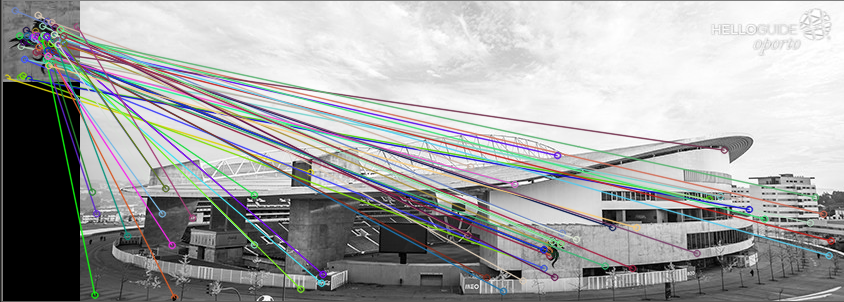
*Fig. 2 - Keypoints da imagem a aumentar*

Depois, por cada ROI guardada na base de dados, deteta os keypoint e extrai os descritores.



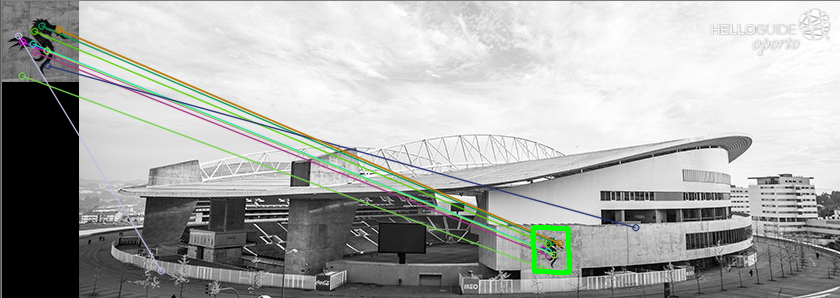
*Fig. 3 – Keypoints da imagem da base de dados*

De seguida faz corresponder todos os keypoints da imagem que pretende aumentar à região guardada na base de dados.

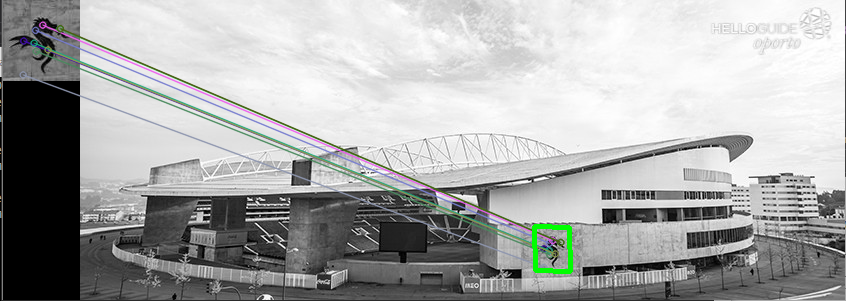


*Fig. 4 – Todas as correspondências*

Devido ao elevado número de correspondências tornou-se necessário reduzir esse mesmo número. De forma a reduzir as correspondências optou-se por se considerar apenas aquelas cujas distâncias entre os keypoints são menores que 3\*distância\_mínima. Se existirem correspondências suficientes é calculada a homografia utilizando o método RANSAC e os inliners (pontos que estão dentro da região de interesse na imagem a aumentar). De seguida são calculados os possíveis cantos da região de interesse na imagem a aumentar. Utilizando os cantos anteriormente calculados são desenhadas linhas entre eles, para delimitar a ROI.



*Fig. 5 – Apenas as melhores correspondências*

Por fim é desenhada na imagem escolhida para ser aumentada a região de interesse que estava guardada na base de dados, resultante do programa de preparação. Importa referir, que só são desenhadas as linhas delimitadoras se todos os inliners estiverem dentro da região delimitada pelos possíveis cantos, anteriormente calculados.

*Fig. 6 – Apenas os inliners*

Como resultado final é mostrada, a imagem escolhida para ser aumentada, a cores com a região de interesse delimitada pelo retângulo azul e por cima e/ou por baixo (dependendo da posição da região) uma etiqueta com o nome da região, que o utilizador definiu no programa de preparação.



*Fig. 7 – Resultado final*

## Resultados

Foram realizados vários testes com diferentes tipos de imagens e diferentes Feature Detector, Descriptor Extractor e Descriptor Matcher.

Depois dos testes realizados chegou-se à conclusão que a combinação SIFT (Feature Detector), SIFT (Descriptor Extractor) e FLANN (Descriptor Matcher) é aquela que permite obter melhores resultados na maior parte das imagens utilizadas.

Em certas imagens só foi possível obter resultados utilizando a combinação FAST – SIFT -FLANN, uma vez que o método FAST permite obter muitos mais keypoints que o método SIFT, aumentando as hipóteses de correspondências.

Também foram obtidos resultados satisfatórios com a combinação SURF-SURF-FLANN.

Independentemente da escolha do Feature Detector e do Descriptor Extractor, não foi possível obter resultados utilizando como Descriptor Matcher o BFMatcher.

## Conclusão

Ao longo do desenvolvimento do programa de preparação surgiu um problema: como permitir ao utilizador escolher as regiões de interesse? Problema esse solucionado, após pesquisa no site da biblioteca OpenCV, com a função selectROIs (descrita anteriormente). Outros pequenos problemas foram solucionados com maior ou menor dificuldade recorrendo ao site da biblioteca OpenCV ou então ao site stackoverflow.com.

De um modo geral a solução proposta encontra-se finalizada no que aos principais objetivos diz respeito. Importa referir que a solução desenvolvida não permite outro método de seleção das ROI’s para além do retângulo, como inicialmente foi proposto através de setas ou legendas de texto.

## Bibliografia

1. <https://docs.opencv.org/2.4/doc/tutorials/features2d/feature_homography/feature_homography.html>
2. <https://docs.opencv.org/2.4/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html>
3. <https://docs.opencv.org/2.4/modules/features2d/doc/common_interfaces_of_descriptor_extractors.html>
4. <https://docs.opencv.org/2.4/modules/features2d/doc/common_interfaces_of_feature_detectors.html>
5. <https://docs.opencv.org/3.3.1/d7/dfc/group__highgui.html#ga0f11fad74a6432b8055fb21621a0f893>
6. <https://stackoverflow.com/>

## Anexos

### A: Código

### Augmentation.h

#pragma once

#include "opencv2\core\core.hpp"

#include "opencv2\features2d\features2d.hpp"

#include "opencv2/calib3d.hpp"

#include "opencv2/xfeatures2d.hpp"

#include <opencv2/xfeatures2d/nonfree.hpp>

using namespace std;

class Augmentation

{

private:

cv::Ptr<cv::FeatureDetector> detector;

cv::Ptr<cv::DescriptorExtractor> extractor;

cv::DescriptorMatcher \*matcher;

string imagePath;

vector<string> imgNames;

vector<string> paths;

bool testMode;

public:

Augmentation();

Augmentation(string i, vector<string> im, string d, string e, string m, vector<string> p, bool t);

~Augmentation();

int init();

vector< cv::DMatch > getGoodMatches(cv::Mat descriptors\_database, cv::Mat descriptors\_scene);

void draw(cv::Mat img, vector<cv::Point2f> scene\_corners, string name);

void debugMode(cv::Mat db\_image, vector<cv::KeyPoint> db\_keypoints, cv::Mat scene, vector<cv::KeyPoint> scene\_key, cv::Mat descriptors\_database, cv::Mat descriptors\_scene, int i);

bool checkInliers(vector<cv::Point2f> inlier\_points, vector<cv::Point2f> corners);

bool openImageAugmentation(const std::string &filename, cv::Mat &image);

};

### Preparation.h

#pragma once

#include <string>

class Preparation

{

private:

std::string image;

std::string savePath;

public:

Preparation(std::string i, std::string p);

Preparation();

~Preparation(void);

int init();

};

### Preparation.cpp

#include "Preparation.h"

#include <opencv2/core.hpp>

#include <opencv2/imgproc.hpp>

#include <opencv2/xfeatures2d.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

#include <direct.h>

using namespace std;

using namespace cv;

using namespace cv::xfeatures2d;

//Constructor

Preparation::Preparation(std::string i, std::string p)

{

this->image = i;

this->savePath = p;

}

//Default Constructor

Preparation::Preparation() {}

//Destructor

Preparation::~Preparation(void) {}

//Read image

bool openImage(const std::string &filename, Mat &image)

{

image = imread(filename);

if (!image.data)

{

cout << endl << "Error reading image " << filename << endl;

return false;

}

return true;

}

//Start preparation

int Preparation::init()

{

Mat image;

vector< Rect > rects;

//Open image

if (!openImage(this->image, image))

{

cout << "Could not open image!" << endl;

return -1;

}

//Select regions of interest

selectROIs("Preparation", image, rects, false, false);

destroyWindow("Preparation");

//Save regions of interest

for (int i = 0; i < rects.size(); i++)

{

Mat roi = image(rects[i]);

stringstream ss;

cout << "Name the ROI number " << i+1 << endl;

string name;

cin >> name;

ss << this->savePath << "\\" << name << ".jpg";

imwrite(ss.str(), roi);

}

waitKey(0);

return 0;

}

### Augmentation.cpp

#include "Augmentation.h"

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

#include "opencv2/imgproc.hpp"

using namespace std;

using namespace cv;

using namespace cv::xfeatures2d;

//Default constructor

Augmentation::Augmentation() {};

//Destructor

Augmentation::~Augmentation() {};

//Constructor

Augmentation::Augmentation(string i, vector<string> im, string d, string e, string m, vector<string> p, bool t)

{

this->imagePath = i;

this->imgNames = im;

this->paths = p;

this->testMode = t;

if (m == "FLANN")

{

this->matcher = new FlannBasedMatcher();

}

else if (m == "BFM")

{

this->matcher = new BFMatcher(NORM\_HAMMING, true);

}

if (d == "FAST")

{

this->detector = FastFeatureDetector::create();

}

else if(d == "SIFT")

{

this->detector = SiftFeatureDetector::create();

}

else if (d == "SURF")

{

this->detector = SurfFeatureDetector::create();

}

else if (d == "ORB")

{

this->detector = ORB::create();

}

if (e == "SIFT")

{

this->extractor = SiftDescriptorExtractor::create();

if (m == "BFM")

{

this->matcher = new BFMatcher(NORM\_L2, false);

}

}

else if(e == "SURF")

{

this->extractor = SurfDescriptorExtractor::create();

if (m == "BFM")

{

this->matcher = new BFMatcher(NORM\_L2, false);

}

}

else if (e == "ORB")

{

this->extractor = ORB::create();

}

else if(e == "BRIEF")

{

this->extractor = BriefDescriptorExtractor::create();

}

else if (e == "FREAK")

{

this->extractor = FREAK::create();

}

}

//Matching descriptors and return the best ones

vector<DMatch> Augmentation::getGoodMatches(Mat descriptors\_database, Mat descriptors\_scene)

{

vector<DMatch> matches;

vector<DMatch> good\_matches;

double max\_dist = 0;

double min\_dist = 100;

//All matches

this->matcher->match(descriptors\_database, descriptors\_scene, matches);

//Quick calculation of max and min distances between keypoints

for (unsigned i = 0; i < matches.size(); i++)

{

double dist = matches[i].distance;

if (dist < min\_dist)

min\_dist = dist;

if (dist > max\_dist)

max\_dist = dist;

}

//Get good matches (i.e. whose distance is less than 3\*min\_dist )

for (unsigned i = 0; i < matches.size(); i++)

{

if (matches[i].distance < 3 \* min\_dist)

{

good\_matches.push\_back(matches[i]);

}

}

return good\_matches;

}

//Open image in color

bool Augmentation::openImageAugmentation(const std::string &filename, Mat &image)

{

image = imread(filename);

if (!image.data)

{

cout << endl << "Error reading image " << filename << endl;

return false;

}

return true;

}

//Draw the region of interest on img

void Augmentation::draw(Mat img, vector <Point2f> scene\_corners, string name)

{

line(img, scene\_corners[0], scene\_corners[1], Scalar(255, 0, 0), 4);

line(img, scene\_corners[1], scene\_corners[2], Scalar(255, 0, 0), 4);

line(img, scene\_corners[2], scene\_corners[3], Scalar(255, 0, 0), 4);

line(img, scene\_corners[3], scene\_corners[0], Scalar(255, 0, 0), 4);

if (img.cols / 2 < scene\_corners[0].y)

{

putText(img, name.erase(name.length() - 4), Point(scene\_corners[3].x, scene\_corners[3].y + 15), 1, 1, Scalar(255, 0, 0), 1, 8, false);

}

else

{

putText(img, name.erase(name.length() - 4), Point(scene\_corners[0].x, scene\_corners[0].y - 15), 1, 1, Scalar(255, 0, 0), 1, 8, false);

}

}

//Mode that shows all steps

void Augmentation::debugMode(Mat db\_image, vector<KeyPoint> db\_keypoints, Mat scene, vector<KeyPoint> scene\_key, Mat descriptors\_database, Mat descriptors\_scene, int i)

{

Mat db\_output, scene\_output, all\_matches;

vector<DMatch> matches;

//Draw keypoints of scene image

cout << "Detect keypoints of scene image." << endl;

drawKeypoints(scene, scene\_key, scene\_output, Scalar::all(-1));

imshow("Scene", scene\_output);

//Draw keypoints of database image

cout << "Detect keypoints of database image: " << this->imgNames[i] << endl;

drawKeypoints(db\_image, db\_keypoints, db\_output, Scalar::all(-1));

imshow("Keypoints of "+this->imgNames[i], db\_output);

waitKey(1);

//Draw all matches

cout << "Matching all keypoints between scene and " << this->imgNames[i] << endl;

this->matcher->match(descriptors\_database, descriptors\_scene, matches);

drawMatches(db\_image, db\_keypoints, scene, scene\_key, matches, all\_matches, Scalar::all(-1), CV\_RGB(255, 255, 255), Mat(), 2);

imshow("All Matches of "+this->imgNames[i], all\_matches);

waitKey(1);

}

//Check if inliners are inside the corners of the possible region of interest

bool Augmentation::checkInliers(vector<Point2f> inlier\_points, vector<Point2f> corners)

{

for (unsigned int i = 0; i < inlier\_points.size(); ++i)

{

if (pointPolygonTest(corners, inlier\_points[i], true) < 0)

{

return false;

}

}

return true;

}

//Init the augmentation program

int Augmentation::init()

{

Mat scene, scene\_gray, result, result\_inliners;

Mat database;

vector<KeyPoint> keypoints\_database, keypoints\_scene;

Mat descriptors\_database, descriptors\_scene;

//Open scene image

if (!openImageAugmentation(this->imagePath, scene))

{

cout << "Could not open image!" << endl;

return -1;

}

//Convert to grayscale

cvtColor(scene, scene\_gray, CV\_BGR2GRAY);

//Detect keypoints of scene image

this->detector->detect(scene\_gray, keypoints\_scene);

//Extract descriptors of scene image

this->extractor->compute(scene\_gray, keypoints\_scene, descriptors\_scene);

//Analyse all images on database

for (int i = 0; i < this->paths.size(); i++)

{

//Open database image

if (!openImageAugmentation(this->paths[i], database))

{

cout << "Could not open image!" << endl;

return -1;

}

//Convert to grayscale

cvtColor(database, database, CV\_BGR2GRAY);

//Detect keypoints of database image

this->detector->detect(database, keypoints\_database);

//Extract descriptors of database image

this->extractor->compute(database, keypoints\_database, descriptors\_database);

//Debug Mode

if (this->testMode)

{

debugMode(database, keypoints\_database, scene\_gray, keypoints\_scene, descriptors\_database, descriptors\_scene,i);

}

//Matching descriptors

vector<DMatch> good\_matches = getGoodMatches(descriptors\_database, descriptors\_scene);

if (good\_matches.size() < 4)

{

cout << "Does not have enough points." << endl;

}

else

{

//Prepare data to findHomography

vector<Point2f> database\_points;

vector<Point2f> scene\_points;

for (size\_t i = 0; i < good\_matches.size(); i++)

{

database\_points.push\_back(keypoints\_database[good\_matches[i].queryIdx].pt);

scene\_points.push\_back(keypoints\_scene[good\_matches[i].trainIdx].pt);

}

// Find homography matrix and get inliers mask

vector<unsigned char> inliersMask;

Mat homography = findHomography(database\_points, scene\_points, RANSAC, 3, inliersMask);

vector<DMatch> inliers;

vector<Point2f> inlier\_points;

for (size\_t i = 0; i<inliersMask.size(); i++)

{

if (inliersMask[i])

{

inliers.push\_back(good\_matches[i]);

inlier\_points.push\_back(keypoints\_scene[good\_matches[i].trainIdx].pt);

}

}

//Apply homography to the object corners position to match the one in the scene

std::vector<Point2f> obj\_corners(4);

std::vector<Point2f> scene\_corners(4);

obj\_corners[0] = cvPoint(0, 0);

obj\_corners[1] = cvPoint(database.cols, 0);

obj\_corners[2] = cvPoint(database.cols, database.rows);

obj\_corners[3] = cvPoint(0, database.rows);

perspectiveTransform(obj\_corners, scene\_corners, homography);

if (this->testMode)

{

//Draw the good matches

drawMatches(database, keypoints\_database, scene\_gray, keypoints\_scene, good\_matches, result, Scalar::all(-1), CV\_RGB(255, 255, 255), Mat(), 2);

line(result, scene\_corners[0] + Point2f(database.cols, 0), scene\_corners[1] + Point2f(database.cols, 0), Scalar(0, 255, 0), 4);

line(result, scene\_corners[1] + Point2f(database.cols, 0), scene\_corners[2] + Point2f(database.cols, 0), Scalar(0, 255, 0), 4);

line(result, scene\_corners[2] + Point2f(database.cols, 0), scene\_corners[3] + Point2f(database.cols, 0), Scalar(0, 255, 0), 4);

line(result, scene\_corners[3] + Point2f(database.cols, 0), scene\_corners[0] + Point2f(database.cols, 0), Scalar(0, 255, 0), 4);

imshow(this->imgNames[i]+" good matches", result);

waitKey(1);

//Draw only the inliners

drawMatches(database, keypoints\_database, scene\_gray, keypoints\_scene, inliers, result\_inliners, Scalar::all(-1), CV\_RGB(255, 255, 255), Mat(), 2);

line(result\_inliners, scene\_corners[0] + Point2f(database.cols, 0), scene\_corners[1] + Point2f(database.cols, 0), Scalar(0, 255, 0), 4);

line(result\_inliners, scene\_corners[1] + Point2f(database.cols, 0), scene\_corners[2] + Point2f(database.cols, 0), Scalar(0, 255, 0), 4);

line(result\_inliners, scene\_corners[2] + Point2f(database.cols, 0), scene\_corners[3] + Point2f(database.cols, 0), Scalar(0, 255, 0), 4);

line(result\_inliners, scene\_corners[3] + Point2f(database.cols, 0), scene\_corners[0] + Point2f(database.cols, 0), Scalar(0, 255, 0), 4);

imshow(this->imgNames[i]+" inliners", result\_inliners);

waitKey(1);

}

//Draw rectangle on scene, only if all inliners are in scene\_corners

if (checkInliers(inlier\_points, scene\_corners))

{

draw(scene, scene\_corners, this->imgNames[i]);

}

}

}

//Show Final Result

imshow("Final", scene);

waitKey(0);

destroyAllWindows();

return 0;

}

### Main.cpp

#include <iostream>

#include <string>

#include "Preparation.h"

#include "Augmentation.h"

#include <direct.h>

#include <vector>

#include <conio.h>

#include <windows.h>

#include <stdio.h>

using namespace std;

//Get full path of folder

string getFullPath(string folder)

{

char \* path = NULL;

path = \_getcwd(NULL, 0);

string fullPath = path + folder;

return fullPath;

}

//Create directory with the name "folder"

string createDirectory(string folder)

{

string fullPath = getFullPath(folder);

const char \* p\_path = fullPath.c\_str();

\_mkdir(p\_path);

return fullPath;

}

//Get all files from path

vector<string> getDirectoryFiles(string path)

{

string pattern(path);

vector<string> temps;

vector<string> names;

pattern.append("\\\*");

WIN32\_FIND\_DATA data;

HANDLE hFind;

if ((hFind = FindFirstFile(pattern.c\_str(), &data)) != INVALID\_HANDLE\_VALUE)

{

do

{

temps.push\_back(data.cFileName);

} while (FindNextFile(hFind, &data) != 0);

FindClose(hFind);

}

for (int i = 0; i < temps.size(); i++)

{

if (temps[i] != ".")

{

if(temps[i] != "..")

names.push\_back(temps[i]);

}

}

return names;

}

//Convert user input into detector name

string getDetector(int opt)

{

if (opt == 1)

{

return "FAST";

}

else if (opt == 2)

{

return "SIFT";

}

else if (opt == 3)

{

return "SURF";

}

else if (opt == 4)

{

return "ORB";

}

}

//Convert user input into extractor name

string getExtractor(int opt)

{

if (opt == 1)

{

return "SIFT";

}

else if (opt == 2)

{

return "SURF";

}

else if (opt == 3)

{

return "ORB";

}

else if (opt == 4)

{

return "BRIEF";

}

else if (opt == 5)

{

return "FREAK";

}

}

//Convert user input into matcher name

string getMatcher(int opt)

{

if (opt == 1)

{

return "FLANN";

}

else if (opt == 2)

{

return "BFM";

}

}

//Convert user input into test mode boolean

bool getMode(int opt)

{

if (opt == 1)

{

return true;

}

else if (opt == 2)

{

return false;

}

}

//Build the path to a file in database

string buildPathToDatabase(string path, string name)

{

string full = path + "\\" + name;

return full;

}

//Build the path to all files in database

vector<string> buildPathToImage(vector<string> names)

{

vector<string> paths;

for (int i = 0; i < names.size(); i++)

{

string path = buildPathToDatabase(getFullPath("\\database"), names[i]);

paths.push\_back(path);

}

return paths;

}

//Delete all files in database

void resetDatabase(vector<string> files)

{

for (int i = 0; i < files.size(); i++)

{

remove(files[i].c\_str());

}

}

int main(int argc, char\*\* argv)

{

int opt;

int detectorOpt;

int extractorOpt;

int matcherOpt;

int testOpt;

string imagePath;

string savePath;

string scenePath;

Preparation p;

Augmentation a;

vector<string> names;

vector<string> paths;

do

{

system("cls");

cout << "================================================================" << endl;

cout << "=========== Augmented Reality with Natural Markers =============" << endl;

cout << "================================================================" << endl << endl;

cout << "Choose one of the following mods:" << endl << endl;

cout << "1. Preparation" << endl;

cout << "2. Augmentation" << endl;

cout << "3. Reset Database" << endl;

cout << "4. Exit" << endl << endl;

cout << "Mode: ";

cin >> opt;

switch (opt)

{

case 1:

system("cls");

savePath = createDirectory("\\database");

cout << "=============================================" << endl;

cout << "============= Preparation Mode: =============" << endl;

cout << "=============================================" << endl << endl;

cout << "Path to Image: ";

cin >> imagePath;

p = Preparation(imagePath, savePath);

p.init();

break;

case 2:

system("cls");

names = getDirectoryFiles(getFullPath("\\database"));

paths = buildPathToImage(names);

cout << "=============================================" << endl;

cout << "============= Augmentation Mode =============" << endl;

cout << "=============================================" << endl;

cout << endl << "Possible Combinations" << endl << endl;

cout << "DETECTOR " << "EXTRACTOR " << "MATCHER" << endl;

cout << "SURF " << "SURF " << "FLANN" << endl;

cout << "SURF " << "FREAK " << "BFM" << endl;

cout << "SURF " << "SURF " << "BFM" << endl;

cout << "SIFT " << "SIFT " << "FLANN" << endl;

cout << "SIFT " << "SURF " << "FLANN" << endl;

cout << "SIFT " << "SURF " << "BFM" << endl;

cout << "FAST " << "SURF " << "FLANN" << endl;

cout << "FAST " << "SIFT " << "FLANN" << endl;

cout << "FAST " << "BRIEF " << "BFM" << endl;

cout << "FAST " << "FREAK " << "BFM" << endl;

cout << "FAST " << "ORB " << "BFM" << endl;

cout << "ORB " << "ORB " << "BFM" << endl;

cout << "ORB " << "BRIEF " << "BFM" << endl << endl;

cout << "Feature Detector" << endl;

cout << "1. FAST" << endl << "2. SIFT" << endl << "3. SURF" << endl << "4. ORB" << endl;

do

{

cout << "Detector: ";

cin >> detectorOpt;

} while (detectorOpt > 4 || detectorOpt <= 0);

system("cls");

cout << "Descriptor Extractor" << endl;

cout << "1. SIFT" << endl << "2. SURF" << endl << "3. ORB" << endl << "4. BRIEF" << endl << "5. FREAK" << endl;

do

{

cout << "Extractor: ";

cin >> extractorOpt;

} while (extractorOpt > 5 || extractorOpt <= 0);

system("cls");

cout << "Descriptor Matcher" << endl;

cout << "1. FLANN" << endl << "2. BFM" << endl;

do

{

cout << "Matcher: ";

cin >> matcherOpt;

} while (matcherOpt > 2 || matcherOpt <= 0);

system("cls");

cout << "Test Mode?" << endl;

cout << "1. Yes" << endl << "2. No" << endl;

do

{

cout << "Mode: ";

cin >> testOpt;

} while (testOpt > 2 || testOpt <= 0);

system("cls");

cout << "Path to Image: ";

cin >> scenePath;

a = Augmentation(scenePath, names, getDetector(detectorOpt), getExtractor(extractorOpt), getMatcher(matcherOpt), paths, getMode(testOpt));

a.init();

system("pause");

break;

case 3:

names = getDirectoryFiles(getFullPath("\\database"));

paths = buildPathToImage(names);

resetDatabase(paths);

system("pause");

break;

default:

break;

}

} while (opt != 4 || opt > 4 || opt < 0);

system("pause");

return 0;

}