Министерство образования и науки Российской федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Алтайский государственный технический университет им. И. И.Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015 г.

Отчет по лабораторной работе №3

Операторы точек интереса

«Интеллектуальные технологии обработки изображений»

ИЗ 231000.3.000 О

Студент группы 8ПИ-41 А.Ю. Смирнова

(И.О. Фамилия)

Преподаватель старший преподаватель М.Г. Казаков

(должность, ученое звание) (И.О. Фамилия)

Барнаул 2015

**Цель работы**

Освоить базовые алгоритмы поиска интересных точек в изображении, которые затем используются для вычисления дескрипторов.

**Формулировка задачи**

* Реализовать операторы Моравека и Харриса для поиска интересных точек в изображении.
* Реализовать фильтрацию интересных точек методом Adaptive Non-Maximum Suppression для заданного количества необходимых точек.
* Оценить повторяемость результата при некоторых искажениях оригинального изображения – сдвиг, шум, контрастность и яркость.

**Исходный код:**

detectors.h

#ifndef DETECTORS\_H

#define DETECTORS\_H

#include "image.h"

#include <vector>

#include <algorithm>

#include <utility>

#include "filtermanager.h"

#include "maskfactory.h"

#include "pyramid.h"

#include "point.h"

class Detectors

{

static vector<Point> findLocalMax(const Image &S, int localMaxSize, int halfSizeW, float bottom);

static float findContrast(const Image &image, int x, int y, int dx, int dy, int halfSizeW);

static float getDistance(Point p1, Point p2);

public:

Detectors();

static vector<Point> Moravec(const Image &image, int halfSizeW, int localMaxSize, float bottom);

static vector<Point> Harris(const Image &image, int halfSizeW, int localMaxSize, float bottom);

static vector<Point> AdaptiveNonMaximumSuppression(const vector<Point> &points, int maxPointNum, int maxR);

~Detectors();

};

#endif

detectors.cpp

#include "detectors.h"

Detectors::Detectors()

{

}

float Detectors::findContrast(const Image &image, int x, int y, int dx, int dy, int halfSizeW)

{

if(dx == dy && dx == 0) return std::numeric\_limits<float>::max();

float sum = 0;

float p1, p2;

for(int i = y - halfSizeW; i<y+halfSizeW; i++)

{

for(int j = x - halfSizeW; j<x+halfSizeW; j++)

{

p1 = image.getPixel(i, j, EdgeMode::MIRROR);

p2 = image.getPixel(i + dy, j + dx, EdgeMode::MIRROR);

sum = sum + pow(p1 - p2, 2);

}

}

return sum;

}

vector<Point> Detectors::Moravec(const Image &image, int halfSizeW, int localMaxSize, float bottom)

{

Image S(image.getHeight(),image.getWidth());

float C;

float min;

for(int i=halfSizeW; i<image.getHeight()-halfSizeW; i++)

{

for(int j=halfSizeW; j<image.getWidth()-halfSizeW; j++)

{

min = std::numeric\_limits<float>::max();

for(int dx = -1; dx<=1; dx++)

{

for(int dy=-1; dy<=1; dy++)

{

C = findContrast(image, j, i, dx,dy, halfSizeW);

if(C < min)

{

min = C;

}

}

}

S.setPixel(i,j,min);

}

}

return findLocalMax(S, localMaxSize, halfSizeW, bottom);

}

vector<Point> Detectors::findLocalMax(const Image &S, int localMaxSize, int halfSizeW, float bottom)

{

vector<Point> points;

bool isLocalMax;

float pixelValue;

for(int i=halfSizeW; i<S.getHeight()-halfSizeW; i++)

{

for(int j=halfSizeW; j<S.getWidth()-halfSizeW; j++)

{

isLocalMax = true;

pixelValue = S.getPixel(i, j);

if(pixelValue >= bottom)

{

for(int dx = -localMaxSize; dx <= localMaxSize; dx++)

{

for(int dy = -localMaxSize; dy <= localMaxSize; dy++)

{

if(pixelValue <= S.getPixel(i + dx, j + dy) && !(dx==0 && dy == 0))

{

isLocalMax = false;

goto exit;

}

}

}

exit: if(isLocalMax)

{

//emplace\_back

points.emplace\_back(j, i, pixelValue,0);

}

}

}

}

return points;

}

vector<Point> Detectors::Harris(const Image &image, int halfSizeW, int localMaxSize, float bottom)

{

auto A = FilterManager::SeparatedFilter(image, \*MaskFactory::SobelSeparated(Asix::X), false, EdgeMode::MIRROR);

auto C = FilterManager::SeparatedFilter(image, \*MaskFactory::SobelSeparated(Asix::Y), false, EdgeMode::MIRROR);

auto B = make\_shared<Image>(image.getHeight(),image.getWidth());

float x,y;

for(int i=0; i<image.getHeight(); i++)

{

for(int j=0; j<image.getWidth(); j++)

{

x = A->getPixel(i,j);

y = C->getPixel(i,j);

A->setPixel(i,j,x\*x);

B->setPixel(i,j,x\*y);

C->setPixel(i,j,y\*y);

}

}

float sigma = halfSizeW \* 1.0 / 3;

A = FilterManager::SeparatedFilter(\*A, \*MaskFactory::GaussSeparated(sigma), false, EdgeMode::MIRROR);

B = FilterManager::SeparatedFilter(\*B, \*MaskFactory::GaussSeparated(sigma), false, EdgeMode::MIRROR);

C = FilterManager::SeparatedFilter(\*C, \*MaskFactory::GaussSeparated(sigma), false, EdgeMode::MIRROR);

//find min lambda

auto L = make\_shared<Image>(image.getHeight(),image.getWidth());

float a,b,c;

float response;

vector<Point> points;

for(int i=halfSizeW; i<image.getHeight()-halfSizeW; i++)

{

for(int j=halfSizeW; j<image.getWidth()-halfSizeW; j++)

{

a = A->getPixel(i,j);

b = B->getPixel(i,j);

c = C->getPixel(i,j);

response = a\*b - c\*c - 0.05\*(a+b)\*(a+b);

L->setPixel(i,j,response);

}

}

return findLocalMax(\*L, localMaxSize, halfSizeW, bottom);

}

float Detectors::getDistance(Point p1, Point p2)

{

return sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));

}

vector<Point> Detectors::AdaptiveNonMaximumSuppression(const vector<Point> &points, int maxPointNum, int maxR)

{

vector<Point> sortedPoints = vector<Point>(points);

if(maxPointNum >= points.size())

{

return sortedPoints;

}

sort(sortedPoints.begin(), sortedPoints.end(), [](auto a, auto b) {return a.contrast > b.contrast;});

vector<Point> result;

vector<bool> isInResult = vector<bool>(sortedPoints.size(), false);

float r = maxR;

bool isMax;

while(result.size() < maxPointNum)

{

r = r/2;

for(uint i=0; i<sortedPoints.size(); i++)

{

if(!isInResult[i])

{

isMax = true;

for(uint j=0; j<sortedPoints.size(); j++)

{

if(sortedPoints[i].contrast\*0.9 < sortedPoints[j].contrast && getDistance(sortedPoints[i], sortedPoints[j]) < r)

{

isMax = false;

break;

}

}

if(isMax)

{

isInResult[i] = true;

result.push\_back(sortedPoints[i]);

if(result.size() == maxPointNum)

{

break;

}

}

}

}

}

return result;

}

**Результаты работы:**

На рисунках представлены результаты работы детектора на оригинальном изображении и изображениях, подвергнутых искажениям – сдвиг, шум, контрастность и яркость.

Результаты работы детектора были обработаны алгоритмом Adaptive Non-Maximum Suppression для получения 50 точек.

Детектор Харриса:





Детектор Моравека:

