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 №4

Дескрипторы окрестности интересных точек

«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 принципы вычисления дескрипторов окрестностей и их представления в виде векторов. Познакомиться с подходом на основе гистограмм градиентов.

**Формулировка задачи:**

* Реализовать вычисление дескрипторов окрестностей заданных точек путем вычисления градиентов в каждой точки изображения и разбиения окрестности на сетку.
* Реализовать вычисление гистограмм градиентов в ячейках сетки и нормализацию полученных дескрипторов.
* Реализовать визуализацию результатов поиска ближайших дескрипторов в двух изображениях.

**Исходый код:**

descriptor.h

#ifndef DESCRIPTOR\_H

#define DESCRIPTOR\_H

#include <memory>

#include <math.h>

#include <vector>

#include <algorithm>

#include <utility>

#include "point.h"

using namespace std;

class Descriptor

{

vector<float> baskets;

public:

Point point;

Descriptor(int basketNum, Point p);

static float distance(Descriptor d1, Descriptor d2);

void normalize();

void addInBasket(int basketNum, float value);

shared\_ptr<Descriptor> findClosest(vector<Descriptor> descriptors);

~Descriptor();

};

#endif

descriptor.cpp

#include "descriptor.h"

Descriptor::Descriptor(int basketNum, Point p)

{

for(int i=0; i<basketNum; i++)

{

baskets.push\_back(0);

}

this->point = p;

}

float Descriptor::distance(Descriptor d1, Descriptor d2)

{

float distance = 0;

for(int i=0; i<d1.baskets.size(); i++)

{

distance += pow(d1.baskets[i] - d2.baskets[i], 2);

}

return sqrt(distance);

}

void Descriptor::normalize()

{

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

float maxValue = 0;

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

{

//baskets[i] = max(0.0f, min(2.0f, baskets[i]));

minValue = min(minValue, baskets[i]);

maxValue = max(maxValue, baskets[i]);

}

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

{

baskets[i] = min((baskets[i] - minValue)/(maxValue - minValue), 0.2f);

}

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

maxValue = 0;

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

{

//baskets[i] = max(0.0f, min(2.0f, baskets[i]));

minValue = min(minValue, baskets[i]);

maxValue = max(maxValue, baskets[i]);

}

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

{

baskets[i] = (baskets[i] - minValue)/(maxValue - minValue);

}

}

void Descriptor::addInBasket(int basketNum, float value)

{

if(basketNum > 0 && basketNum < (int)baskets.size())

{

baskets[basketNum] += value;

}

}

shared\_ptr<Descriptor> Descriptor::findClosest(vector<Descriptor> descriptors)

{

sort(begin(descriptors), end(descriptors), [this](auto a, auto b) {return distance(\*this,a) < distance(\*this,b);});

if(distance(\*this, descriptors[0]) / distance(\*this, descriptors[1]) < 0.8)

{

return make\_shared<Descriptor>(descriptors[0]);

}

return nullptr;

}

Descriptor::~Descriptor()

{

}

descriptorgenerator.h

#ifndef DESCRIPTORGENERATOR\_H

#define DESCRIPTORGENERATOR\_H

#include "image.h"

#include "filtermanager.h"

#include "maskfactory.h"

#include "descriptor.h"

#include <vector>

using namespace std;

class DescriptorGenerator

{

unique\_ptr<Image> gradients;

unique\_ptr<Image> angles;

public:

DescriptorGenerator(const Image& image);

Descriptor getDescriptor(Point p, int surSize, int gistNum, int basketNum);

~DescriptorGenerator();

};

#endif

descriptorgenerator.cpp

#include "descriptorgenerator.h"

DescriptorGenerator::DescriptorGenerator(const Image& image)

{

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

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

gradients = make\_unique<Image>(image.getHeight(), image.getWidth());

angles = make\_unique<Image>(image.getHeight(), image.getWidth());

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

{

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

{

angles->setPixel(i,j,atan2(dy->getPixel(i,j), dx->getPixel(i, j))\*180/M\_PI + 180);

gradients->setPixel(i, j, sqrt(pow(dx->getPixel(i,j),2) + pow(dx->getPixel(i,j),2)));

}

}

}

Descriptor DescriptorGenerator::getDescriptor(Point p, int surSize, int gistNum, int basketNum)

{

Descriptor descriptor(basketNum\*gistNum, p);

int gistSize = ceil(surSize/gistNum);

int curGistNum;

float weight,angle;

float oneBasket = 360 / basketNum;

int left;

float leftValue;

int x = p.x - surSize/2;

int y = p.y - surSize/2;

float sigma = surSize\*0.5;

int x1,y1;

for(int i=0; i<gistNum; i++)

{

for(int j=0; j<gistNum; j++)

{

curGistNum = i\*gistNum+j;

for(int curX = i\*gistSize; curX<(i+1)\*gistSize && curX<surSize; curX++)

{

for(int curY = j\*gistSize; curY<(j+1)\*gistSize && curY<surSize; curY++)

{

x1 = curX -surSize/2;

y1 = curY - surSize/2;

weight = gradients->getPixel(y+curY, x+curX)\*(pow(M\_E,-(x1\*x1 + y1\*y1)/(2\*sigma\*sigma)))/(2\*M\_PI\*sigma\*sigma);//mask->getPixel(curY, curX)\*gradients->getPixel(y+curY, x+curX);

angle = angles->getPixel(y+curY, x+curX);

left = angle/oneBasket;

leftValue = weight\*(angle - left\*oneBasket)/(oneBasket);

descriptor.addInBasket(curGistNum\*basketNum + left, leftValue);

descriptor.addInBasket(curGistNum\*basketNum + left + 1, weight - leftValue);

}

}

}

}

descriptor.normalize();

return descriptor;

}

DescriptorGenerator::~DescriptorGenerator()

{

}

**Результаты работы программы:**

На рисунках показан поиск ближайших дескрипторов двух изображений. Второе изображение получено на основе первое путем сдвига, поворота, масштабирования и афинного преобразования. Видно, что полученные дескрипторы обладают инвариантностью к сдвигу, но не к другим искажениям.

****

