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 №5

Инвариантность дескрипторов к вращению

«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**

Познакомиться с возможностью достижения относительной инвариантности дескрипторов к вращению изображений.

**Формулировка задачи**

* Реализовать относительную инвариантность вычисления дескрипторов относительно вращения изображений на основе подхода SIFT.
* Реализовать этап оценки ориентации интересной точки и поворота сетки, в которой вычисляются гистограммы градиентов.
* Оценить полученный алгоритм с точки зрения реакции на соответствующие искажения изображений, сравнить с полученным в четвертой работе.

**Исходый код:**

descriptorgenerator.cpp

shared\_ptr<Descriptor> DescriptorGenerator::getAngleDescriptor(Point p, int surSize)

{

int beansNum = 48;

int gistNum = 1;

auto descriptor = make\_shared<Descriptor>(beansNum\*gistNum, p);

surSize = surSize\*p.innerScale;

int gistSize = ceil(surSize/gistNum);

int curGistNum;

float weight,angle,r;

float oneBean = 360 / beansNum;

int first, second;

float firstValue, secondValue;

float sigma = surSize\*0.5;

for(int i = -surSize/2; i < surSize/2; i++)

for(int j = -surSize/2; j < surSize/2; j++)

{

weight = magnitudes->getPixel(p.y+i, p.x+j)\*(pow(M\_E,-(i\*i + j\*j)/(2\*sigma\*sigma)))/(2\*M\_PI\*sigma\*sigma);

angle = angles->getPixel(p.y+i, p.x+j);

curGistNum = ((i + surSize/2) / gistSize) \* gistNum + (j + surSize/2) / gistSize;

descriptor->addInGist(curGistNum, angle, weight, beansNum);

}

return descriptor;

}

shared\_ptr<Descriptor> DescriptorGenerator::getDescriptor(Point p, int surSize, int gistNum, int beansNum)

{

//find point angle

auto descriptor = make\_shared<Descriptor>(beansNum\*gistNum, p);

surSize = surSize\*p.innerScale;

int gistSize = ceil(surSize/gistNum);

int curGistNum;

float weight,angle,r;

float oneBean = 360 / beansNum;

int first, second;

float firstValue, secondValue;

float sigma = surSize\*0.5;

float dx, dy;

float radA = p.angle \* M\_PI / 180;

int curGistX, curGistY;

float topLeftCenterX, topLeftCenterY;

for(int i = -surSize/2 - sqrt(2); i < surSize/2 + sqrt(2); i++)

for(int j = -surSize/2 - sqrt(2); j < surSize/2 + sqrt(2); j++)

{

weight = magnitudes->getPixel(p.y+i, p.x+j)\*(pow(M\_E,-(i\*i + j\*j)/(2\*sigma\*sigma)))/(2\*M\_PI\*sigma\*sigma);

angle = - p.angle + angles->getPixel(p.y+i, p.x+j);

while(angle < 0) angle += 360;

dx = j \* cos(radA) + i\*sin(radA);

dy = - j \* sin(radA) + i\*cos(radA);

curGistX = (int)(dy + surSize/2) / gistSize;

curGistY = (int)(dx + surSize/2) / gistSize;

curGistNum = (curGistY)\*gistNum + curGistX;

descriptor->addInGist(curGistNum, angle, weight, beansNum);

}

descriptor->normalize();

return descriptor;

}

descriptor.cpp

float Descriptor::getAngle(int bean)

{

float oneBean = 360.0/beansNum;

int left = (bean - 1 + beansNum) % beansNum;

int right = (bean + 1) % beansNum;

float y2 = beans[bean];

float x2 = bean \* oneBean + oneBean/2;

float y1 = beans[left];

float x1 = x2 - oneBean;

float y3 = beans[right];

float x3 = x2 + oneBean;

float a = (y3 - (x3\*(y2-y1)+x2\*y1-x1\*y2)/(x2-x1)) / (x3\*(x3-x1-x2) + x1\*x2);

float b = -(a\*(x1+x2)) +((y2 - y1)/(x2 - x1));

return -b/(2\*a);

}

pair<float,float> Descriptor::getMaxAngle()

{

int maxIndex1 = -1, maxIndex2 = -1;

float maxValue1 = numeric\_limits<float>::min(),

maxValue2 = numeric\_limits<float>::min();

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

{

if(beans[i] > maxValue1)

{

maxValue2 = maxValue1;

maxIndex2 = maxIndex1;

maxValue1 = beans[i];

maxIndex1 = i;

}

else if(beans[i] > maxValue2)

{

maxValue2 = beans[i];

maxIndex2 = i;

}

}

if(maxValue2 / maxValue1 > 0.8)

{

return make\_pair<float, float>(getAngle(maxIndex1), getAngle(maxIndex2));

}

return make\_pair<float, float>(getAngle(maxIndex1), numeric\_limits<float>::min());

}

void Descriptor::addInGist(int curGistNum, float angle, float weight, int beanInGist)

{

float oneBean = 360.0 / beanInGist;

int first = angle/oneBean;

int second;

if(angle < first\*oneBean + 0.5\*oneBean)

{

second = first - 1;

if(second < 0) second += beanInGist;

}

else

{

second = (first + 1)%beanInGist;

}

float r = fabs(first\*oneBean + 0.5\*oneBean - angle);

float firstValue = weight\*(oneBean - r)/oneBean;

float secondValue = weight - firstValue;

addInBean(curGistNum\*beanInGist + first, firstValue);

addInBean(curGistNum\*beanInGist + second, secondValue);

}

**Результаты работы программы:**

