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 №9

Оценка трансформации через преобразование Хафа

«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 метод преобразования Хафа для поиска модели трансформации между изображениями.

**Формулировка задачи**

* Даны два изображения, причем на втором из них помещен объект из первого изображения, возможно с измененным масштабом и/или углом поворота.
* С помощью извлеченного набора дескрипторов из обоих изображений и их соответствий, найти объект из первого изображения на втором, получить соответствующую позу.
* Продемонстрировать работу метода с различными настройками квантования фазового пространства и визуализацией полученных поз объекта на втором изображении.
* (усложнение) Несколько искомых объектов и несколько экземпляров в анализируемом изображении.

**Исходный код:**

models.h

#ifndef MODELS\_H

#define MODELS\_H

#include "image.h"

#include "descriptor.h"

#include "modelparameter.h"

#include <gsl/gsl\_linalg.h>

#include <gsl/gsl\_blas.h>

class Models

{

vector<pair<shared\_ptr<Descriptor>, shared\_ptr<Descriptor>>> matches;

public:

vector<pair<Point, Point>> best;

double bestModel[9];

Models(vector<shared\_ptr<Descriptor>> desc1, vector<shared\_ptr<Descriptor>> desc2);

double \*RanSaC(int iterCount, float eps);

double \*Hough(float eps);

~Models();

};

#endif

models.cpp

double \*Models::Hough(float eps)

{

//дескрипторы искомого объекта - первые в паре

map<ModelParameter, vector<int>> votes;

double x, y, angle, scale;

Point point1, point2;

int xBin, yBin, aBin, sBin;

//Lenna

// double dx = 2, dy = 2, dangle = 3.14/7.5, dscale = 0.25;

// int shareBin = 3;

//Box

double dx = 1, dy = 1, dangle = 3.14/16, dscale = 0.25;

int shareBin = 2;

//строим четырехмерный аккумулятор

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

{

point1 = matches[i].first->point;

point2 = matches[i].second->point;

scale = point2.scale / point1.scale;

angle = (point2.angle - point1.angle)\*3.14/180;

//while(angle < 0) angle += 2\*3.14;

//while(angle > 2\*3.14) angle -= 2\*3.14;

x = point1.x \* cos(angle) - point1.y \* sin(angle);

y = point1.x \* sin(angle) + point1.y \* cos(angle);

x = point2.x - x\*scale;

y = point2.y - y\*scale;

xBin = round(x / dx);

yBin = round(y / dy);

aBin = round(angle / dangle);

sBin = round(scale / dscale);

for(int sx = -shareBin; sx <=shareBin; sx++)

{

for(int sy = -shareBin; sy <=shareBin; sy++)

{

for(int sa = -shareBin; sa <=shareBin; sa++)

{

for(int ss = -shareBin; ss <=shareBin; ss++)

{

auto model = ModelParameter(xBin + sx, yBin + sy, sBin + ss, aBin + sa);

if(votes.find(model) == votes.end())

{

votes[model] = vector<int>();

}

if(find(begin(votes[model]), end(votes[model]), i) == votes[model].end())

{

votes[model].push\_back(i);

}

}

}

}

}

}

double h[9];

int inliers;

double xInit, yInit, xExpect, yExpect, xCur, yCur;

int bestInliers = -1;

gsl\_matrix \*A = gsl\_matrix\_alloc(6,6);

gsl\_vector \*b = gsl\_vector\_alloc(6);

gsl\_vector \*solution = gsl\_vector\_alloc(6);

gsl\_permutation \* p = gsl\_permutation\_alloc (6);

int signum;

int best;

for(auto vote = votes.begin(); vote != votes.end(); vote++)

{

if(vote->second.size() > 2)

{

//qDebug() << vote->second.size();

for(int i=0; i<3; i++)

{

xInit = matches[vote->second[i]].first->point.x;

yInit = matches[vote->second[i]].first->point.y;

xExpect = matches[vote->second[i]].second->point.x;

yExpect = matches[vote->second[i]].second->point.y;

gsl\_matrix\_set(A, i\*2, 0, xInit);

gsl\_matrix\_set(A, i\*2, 1, yInit);

gsl\_matrix\_set(A, i\*2, 2, 1);

gsl\_matrix\_set(A, i\*2 + 1, 3, xInit);

gsl\_matrix\_set(A, i\*2 + 1, 4, yInit);

gsl\_matrix\_set(A, i\*2 + 1, 5, 1);

gsl\_vector\_set(b, i\*2, xExpect);

gsl\_vector\_set(b, i\*2 + 1, yExpect);

}

gsl\_linalg\_LU\_decomp (A, p, &signum);

gsl\_linalg\_LU\_solve (A, p, b, solution);

for(int i=0; i<6; i++)

{

h[i] = gsl\_vector\_get(solution, i);

}

h[6] = 0;

h[7] = 0;

h[8] = 1;

//find inliners count

inliers = 0;

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

{

xInit = matches[i].first->point.x;

yInit = matches[i].first->point.y;

xExpect = matches[i].second->point.x;

yExpect = matches[i].second->point.y;

xCur = (h[0]\*xInit + h[1]\*yInit + h[2]) / (h[6]\*xInit + h[7]\*yInit + h[8]);

yCur = (h[3]\*xInit + h[4]\*yInit + h[5]) / (h[6]\*xInit + h[7]\*yInit + h[8]);

if(sqrt(pow(xExpect - xCur,2)+pow(yExpect - yCur,2)) < eps)

{

inliers++;

}

}

if(inliers > bestInliers)

{

for(int i=0; i<9; i++)

{

bestModel[i] = h[i];

}

bestInliers = inliers;

}

}

}

gsl\_matrix\_free(A);

gsl\_vector\_free(b);

gsl\_vector\_free(solution);

gsl\_permutation\_free(p);

qDebug() << "Model:";

for(int i=0; i<9; i++)

{

qDebug() << bestModel[i];

}

qDebug() <<"Inliers: " << bestInliers;

return bestModel;

}

Models::~Models()

{

}

mainwindow.cpp

QImage MainWindow::drawObject(const Image &img1, const Image& img2, double \* h)

{

QImage result = QImage(img1.getWidth(), img1.getHeight(), QImage::Format\_RGB32);

result.fill(0);

QPainter painter(&result);

painter.drawImage(0,0,img1.toQImage());

painter.setPen(QPen(QColor(Qt::red)));

QPointF lines[5];

lines[0] = QPointF(h[2], h[5]);

lines[1] = QPointF(h[0]\*(img2.getWidth() - 1) + h[2], h[3]\*(img2.getWidth() - 1) + h[5]);

lines[2] = QPointF(h[0]\*(img2.getWidth() - 1) + h[1]\*(img2.getHeight() - 1) + h[2],h[3]\*(img2.getWidth() - 1) + h[4]\*(img2.getHeight() - 1) + h[5]);

lines[3] = QPointF(h[1]\*(img2.getHeight() - 1) + h[2], h[4]\*(img2.getHeight() - 1) + h[5]);

lines[4] = QPointF(h[2], h[5]);

painter.drawPolyline(lines, 5);

painter.end();

return result;

}

modelparameter.h

#ifndef MODELPARAMETER\_H

#define MODELPARAMETER\_H

struct ModelParameter

{

float x;

float y;

float scale;

float angle;

public:

ModelParameter(float x, float y, float scale, float angle) : x(x), y(y), scale(scale), angle(angle){}

~ModelParameter(){}

bool operator< (const ModelParameter &p) const {

if (scale != p.scale)

return scale < p.scale;

if (angle != p.angle)

return angle < p.angle;

if (x != p.x)

return x < p.x;

return y < p.y;

}

};

#endif

**Результаты работы программы:**

Исходные изображения:



Результат:



Исходные изображения:

 

Результат:

